uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Utpal Acharjee
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)
GRADE / DEGREE

Systems Science
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Personalized and Artificial Intelligence Web Catching and Prefetching

TITRE DE LA THÈSE / TITLE OF THESIS

Professor A. El Saddik
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Professor Wail Gueaieb

Professor Amiya Nayak

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Personalized and Artificial Intelligence Web Caching and Prefetching

By

Utpal Acharjee

A Master's thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of

Master
Of
Systems Science

Systems Science
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-18388-5
Our file Notre référence
ISBN: 978-0-494-18388-5

Abstract

Web caching and prefetching are the most popular and widely used solutions to remedy Internet performance problems. Performance is increased if a combination of caching and prefetching systems is used rather than if these techniques are used individually. Web caching reduces the bandwidth consumption and network latency by serving the user's request from its own cache instead of the original Internet source. Prefetching is a technique that preloads and caches the web object that is not currently requested by the user but can be requested (expected) in the near future. It provides low retrieval latency for users and as well as high hit ratios. Existing methods for caching and prefetching are mostly traditional sharable Proxy cache servers.

In our personalized caching and prefetching approach, the system builds up a user profile associated with a user's web behaviour by parsing the keywords from HTML pages that are browsed by the user. The keywords of a user profile are updated by adding a new keyword or incrementing its associated weight if it is already in the profile. This user profile reflects users' web behaviour or interest. In this cache and prefetch prediction module we considered both static and dynamic users' web behaviour. We have designed and implemented an artificial intelligence multilayer neural network-based caching and prediction algorithm to personalize the Proteus Proxy server with this mechanism. Enhanced Proteus is a multilingual and internationally-supported Proxy system and can work with both mobile and traditional Proxy server-based sharable environments. In the prefetch option of Proteus, we also implemented a unique content filtering feature that blocks the downloading of unwanted web objects.

Acknowledgements

I have sincere gratitude for my academic supervisor Dr. Abdulmotaleb El Saddik who helped, encouraged, and guided me towards not only my education but also my personal success. This thesis would not be successful without his significant contributions. Throughout my work he provided me with a substantial amount of assistance and helpful suggestions. I would also like to sincerely thank the fellow students of the Multimedia Communication Research Laboratory team, especially Bogodan Solomon and Md. Abdur Rahman, for many helpful discussions and constant encouragement and support.

I would also like to thank my beloved parents and my wife for their support throughout my graduate studies. I am also grateful to all my friends and family who helped me with suggestions and positive encouragement. Finally, my love and gratitude are extended to our kids Rttik, Ridoy and Upama. Without their love, understanding, and support this work would never have been finished.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADSL: Asymmetric Digital Subscriber Line

AIMNN: Artificial Multilayer Neural Network

ANN: Artificial Neural Network

API: Application Programming Interface

AWT: Abstract Window Toolkit

DHCP: Dynamic Host Configuration Protocol

DUP: Dynamic User Profile

GUI: Graphical User Interface

HTML: Hyper Text Markup Language

HTTP: Hyper Text Transfer Protocol

HTTPS: Secure Hyper Text Transfer Protocol

IE: Internet Explorer

I/O: Input/Output

IP: Internet Protocol

ISP: Internet Service Provider

JDK: Java Development Kit

JVM: Java Virtual Machine

LAN: Local Area Network

MLP: Multilayer Perceptron

NAT: Network Address Translation

NTP: Network Time Protocol

OS: Operating System

QoS: Quality of Service

SDK: Standard Development Kit

SUP: Static User Profile

SWT: Standard Widget Toolkit

TCP: Transmission Control Protocol

TTL: Time-To-Live

UDP: User Datagram Protocol

UI: User Interface

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

XML: eXtensible Markup Language

Chapter 1: Introduction

1.1 Thesis Background and Motivation

The Internet is the fastest growing technological industry and resource for information. It has become a popular open-text library of information due to being inexpensive and having easily accessible resources. The exponential growth of web servers, services and Internet users generate enormous traffic on the Internet. This traffic creates delays and congestion problems when accessing the web to retrieve information. To overcome this problem, there are several techniques available at the hardware and software levels. The hardware-based solutions improve the Internet bandwidth and device latencies but the user's expectations never end [14].

The most popular software based solutions are web caching and prefetching technologies. Caching and prefetching can work individually or combined. The combination of caching and prefetching enables doubling the performance compared to single caching [27]. These two techniques are very useful tools to reduce congestion, delays and latency problems. The three most important features of web caching are: (i) caching that reduces network bandwidth usage, (ii) caching that also reduces user-perceived delays, and finally (iii) caching that reduce loads on the original server [17].

It is proven that web cache technology improves the Internet response time by providing web content to the requester from its own local cache with minimum interruptions to the original server and network. As a result, this minimum bandwidth usage reduces loads on the original server and improves the user's perceived delays. The Proxy cache server acts as an original web server and function similarly to the original web server. Its main objective is to store the web object to prevent the same object from downloading repeatedly for the requester over the network. As a result, the response time, the load on the network and the load on the original server are all reduced. Previous research shows that the caching objects are declining [5], [7] due to new technology such as dynamic pages and dynamic contents.

Prefetching is another very important technique; it is utilized to preload the web objects that are not yet requested by users but are expected to be requested in the near future. Prefetching is not a technique used to download the user's requested object (cache service performs this action); prefetching downloads the probabilistic pages (preload) that could soon be requested by the user. We found that in many existing web caching and prefetching studies combinations of both have increased the performance results [13], [27]. It is important to take into consideration the impact of these two techniques combined with what we studied in our research. In our study, we found that web caching and prefetching complement each other and provide a better solution than when used individually. According to J. C. Mogul [27], web caching and prefetching combined together can potentially improve latency by up to 60%, whereas simply caching improves the latency by a minimal 26%.

For optimal performance of these two techniques it is essential to select or design an adequate caching strategy, prefetch prediction strategy and cache replacement strategy. There are a number of popular caching and cache-replacement strategies for caching and cache-replacement policies such as Least Recently Used (LRU), Least Frequently Used (LFU), SIZE, and Greedy-Dual-Size etc. There are also a number of prefetching strategies that can be found for the prediction algorithm [1] [25] [35] [37]. There are two basic types of prediction mechanisms available in research and industry including history-based prediction and content-based prediction. History-based predictions, such as [15], [38], are based on the user's history of web action. The content-based prediction uses contextual clues based on the link text which is studied by researcher [1], [37], and others. The history-based prediction system has some limitations; it is unable to prefetch those objects whose URLs have never been visited before. Context-based prediction does not depend on the visited URLs; it depends on the context of the link. The contents of an anchor text of recently visited links or pages suiting the user's interest and the contents of the links or pages significantly influences the future selection of a new link [17], even it is not visited before. This means the user's current web interests may reflect his or her future interests.

Most of the above caching and prefetching systems reside on sharable Proxy servers that allow users that are located behind the Proxy server to share these cached objects. In most cases, the caching and prefetching system run on the same Proxy server, usually located at the gateways of Internet. This is a challenge for mobile users, who frequently move from network to network. Since the prefetching predictions are based on the history of the network-user's behaviour, these predictions cannot carry out the expected results of mobile users since his or her history is at the static server. Moreover, most of the prediction algorithms are based on the history of web action. Most of the popular prefetching algorithms don't have any proper control over prefetching systems; as a result, preloads a number of unwanted documents that cause traffic on the network and fill up the cache with unnecessary objects. For this work, we developed and implemented a client-based personalized Artificial Intelligence Multilayer Neural Network (AIMNN)-based caching and prefetching system Proteus, which resolves the mobile users' caching and prefetching problems, filters unwanted objects and controls the preloading operations. Both caching and prefetching strategies of Proteus are based on the user's static and dynamic web behaviour.

In this research project, we present a brief study of caching and prefetching techniques and their performance in different scenarios including caching and prefetching working together and working individually. In this research, we also analyze user behaviour on the web; analyze users' content access patterns; make estimations of the total average latency, cache hit ratio, successful predictions - the number of predicted objects that are requested immediately after prefetching; unsuccessful predictions - the number of web objects that are not requested by the user immediately after prefetching; prefetch hits - the number of prefetch object that are subsequently demanded by the user; objects Not Used - the total number of prefetch objects that were not demanded by using the access, cache and prefetch logs

1.2 Thesis Objective and Contribution

Users' satisfaction is the key in an industry like the Internet. Users desire quick and appropriate responses from the Internet. A study [43] shows in their "eight second rule" that users won't return to a site that takes more than eight seconds to respond and will leave the site in frustration. Thus, access time is a big challenge for this fast- growing industry as well as web accessibility.

Prefetching, which preloads users' expected web objects, adds an additional value to web caching, improving web performance. A "Web Latency Reduction Study shows that performance is doubled when web caching and prefetching work together rather than just caching alone. Prefetching prediction is a complex element since a wrong prediction has greater costs than benefits. A recent study shows that a number of algorithms are available to predict user actions on the web. Most of the algorithms rely on the prediction of the users' past history by assessing the user's web access log. History-based prefetching systems only predict documents whose URLs were visited previously, meaning that this technique does not work if the link has never been visited. Context-based prediction algorithms are not dependent on visited URLs; they depend on the context of the link meaning that these links could be prefetched even though they have never been visited before.

The main objective of this thesis is to enhance and optimize typical context (keyword) -based caching and prefetching environments with an artificial intelligence neural network algorithm. The system should work for both mobile and immobile environments. This prefetching algorithm should also have some new functions including content filtering, automated user profile management and as well as intelligence prefetching and cache management. Another objective of this thesis is to implement an intelligence caching and cache removal strategy that are based on the user web behaviours or interests. Finally, we will integrate intelligence caching and prefetching techniques together and study the performance of the integrated system. In this study, we emphasize prefetching technology, which is also artificial intelligence.

Our major contribution in this thesis is to design, develop and implement a learning-based, personalized caching and prefetching system for mobile and immobile users; the system is based on user's web interest by considering their static and dynamic behaviour. The main feature of this system is to personalize an artificial intelligence caching and prefetching system for mobile users. Promoting a roaming caching profile, another unique feature we added to this project is content filtering; meaning unwanted web objects are blocked by this system even if they are cacheable or prefetchable. To improve the prefetch prediction, we implemented an algorithm by using an Artificial Intelligence Multilayer Neural Network (AIMNN).

Cache management policy is an important challenge for personalized caching and prefetching systems due to the limited mobile device resources. Not only are the resources an issue but also the web contents and the technologies are changing quickly which creates an extra challenge for cache management. It is essential to use these resources properly. To make proper use of the resources, we studied and implemented an artificial intelligence neural network-based cache management system that includes caching and cache replacement strategy by considering the user's web interests, the web object's cacheable properties e.g. object type, and object size and age.

The system is also designed to support various languages. This system is very flexible and can be converted from a personalized to a sharable cache and prefetch server using minimum effort.

1.3 Thesis Organization

Chapter 2 contains the thesis background and work concerning the development of the web caching and prefetching systems. In chapter 3, the Proteus system design is presented in the goal of system architecture and software design. Chapter 4 provides the implementation details of Proteus while chapter 5 provides the measurement data we

found along with a comparison of features to other tools that are mentioned in chapter 2. In chapter 6, a conclusion is presented and future work is mentioned.

Chapter 2: Background and Related work

2.1 Literature Background

This section presents the background information for the thesis. We describe the concepts of web caching and prefetching technologies. Web caching and prefetching addresses the issue of capacity and performance of web engineering. Caching can increase the performance of web services by transferring the workload from the original server to the nearby Proxy cache server. Prefetching predicts future web activity and preloads the predictable web content for the user. These two techniques satisfy client requests by serving the requested object from its local or nearest cache, regardless of the original server location.

2.1.1 The Web

The web is a simple client server technology consisting of a web server that accepts the requests from the web client for web objects. The requests from the client and the response from the server run on a standard protocol called Hypertext Transfer Protocol (HTTP). The web objects are either stored in a server side with a standard format or follow some kind of relationship so that a user can easily read them, navigate them and find other objects easily. The relationships between web objects assist users in downloading objects and their embedded objects as well as users who can navigate from one object to another by using navigation buttons. This relationship between objects is called a link. Each object is unique identified by the Uniform Resource Locator (URL). This is a global address for web objects and other resources on the World Wide Web (WWW). This address has two parts; the first part includes the protocols and the second indicates the understood human domain name or IP address. In the next few sections, we will discuss different technologies and concepts related to the web.

2.1.2 Web Caching

As we discussed in the previous section, Internet response times can be improved by improving the bandwidth and decreasing the network load. The bandwidth improvement is not always a good solution due to the cost of hardware and some other issues (location, network infrastructure, environment, etc.). The second and most popular approach is web caching, otherwise known as Proxy caching, which decreases the network load by providing the user requested web content from its local storage. In this approach, the Proxy system transfers and stores the web content with the client, which is used repeatedly. The technology of caching effectively transfers popular web documents from web servers to the nearest storage to the client's site's nearest storage. The client's request is addressed by the cache server because the web content is already transferred in it's local cache. In Figure 2.1, the client's request is satisfied by the original server through the Proxy cache server. The Proxy server stores a copy of the requested web content during content delivery to the client. Figure 2.2 shows the client request being addressed by the Proxy cache server by providing web content from its own storage with minimal interruptions.

There are two main advantages to using web cache: reduction of the web latency and the reduction of internet traffic. Web latency is defined by computing the time difference between users' request and Internet response time. The user faces this problem due to the DNS query, creating and establishing a connection for communications and transmitting the requests and responses between clients and servers as well as some server processing times. Latency is reduced because the request is satisfied from the nearest cache and Internet traffic is reduced because the objects are already cached into the Proxy server, meaning clients' requests are not passed to the original server. The caching ability of a web cache server provides exactly the same functionality as an original server.

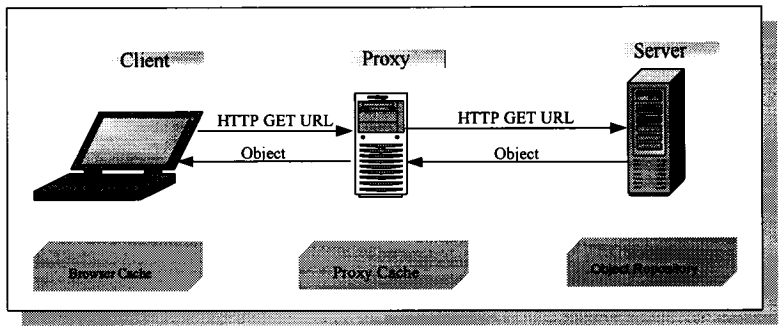

Figure 2.1: Client request satisfied by the original server via Proxy

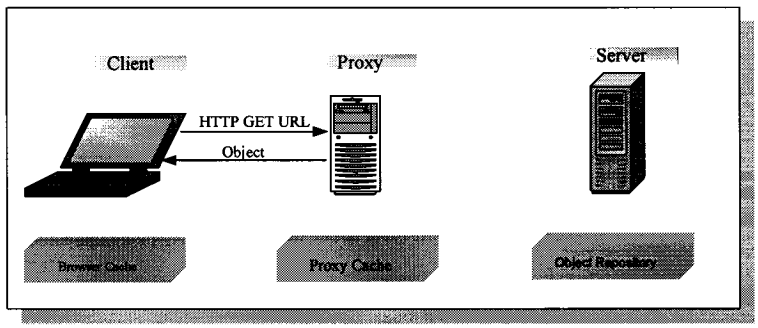

Figure 2.2: Client request satisfied by the Proxy cache server

If the request is satisfied by the Proxy cache server when a client sends an http request as shown in Figure 2.2, the cache server will return the web object to the client from its own cache, the cache hit. If the request is not satisfied, then the Proxy server will forward the request to the original server. The original server addresses the request and calls it a miss hit. In a miss hit situation, the http response will return with a web object to the Proxy cache server and then the Proxy cache server will relay the object to the requester. During this session, the Proxy server saves a copy of this object in its cache, which will be served for the next request.

2.1.2.1 Basic architecture of web caching

Figure 2.3 shows the basic architecture of a web caching system. The Proxy caching system stores the recently visited web content in its cache system. Normally the Proxy server resides on the edge of the corporate network as a shared resource.

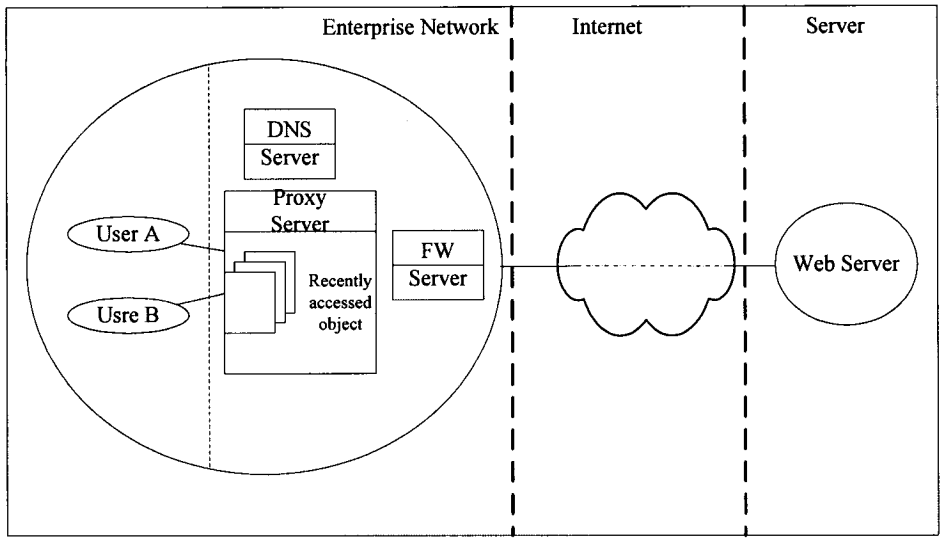

Figure 2.3: Basic architecture of web caching.

When a client requests a web resource, the response goes through a process as shown in the above diagram. At first the Domain Name System (DNS) resolves the name to the IP address and then the client establishes a connection to either a Proxy server or the original server via Proxy. Once the original server receives a request, it analyzes the request and responds to the client with the appropriate header information or data via the Proxy server. The Proxy server stores a copy of the web object (cached) while serving to the client. The whole process of this request response requires considerable time as demonstrated in Figure 2.4.

If the user's request is found to be a miss-hit by the cache server, only then will the request be forwarded to the original server. The original server responds to the client with the requested web content via the Proxy server. If the objects are satisfied in the caching strategy, the Proxy cache server adds the objects to the cache storage.

2.1.2.2 Hypertext Transfer Protocol (HTTP)

HTTP is a client-server model protocol; i.e. it works at the application level. HTTP is used in many aspects such as distributive, collaborative, and hyper media information systems. We will discuss some important features that are web-cache related in the next

few sections. The HTTP protocols are enhanced depending on what industry or researcher require. The first version of HTTP (0.9) was the first widely accepted version. It was then modified to HTTP (1.0) [8] and then HTTP (1.1) [18]. The interaction between the client, the server and others are specified in this HTTP protocol.

The HTTP request and response protocol runs on top of the network protocol TCP for connection and data transfer. There exists two performance optimizations; persistent connections that allow multiple web interaction to reuse the same TCP connection; and pipelining, which allows a client to send a request without waiting for the full response from the previous request[31]. The two HTTP main features, request and response, will be discussed in the following sections.

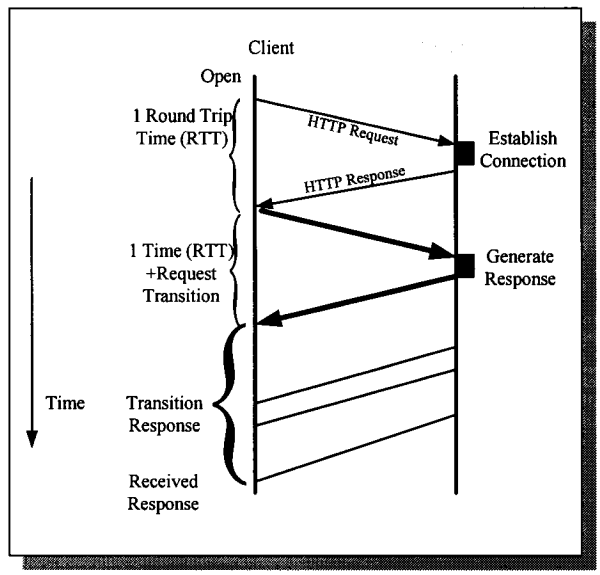

Figure 2.4: Timing cost for a HTTP connection.

As shown in the above diagram, there is some delay for the connection establishment and response between server and client. The new connection for retrieving web objects requires some time, which is approximately two round trip times.

HTTP Request

A basic layout of a request message is shown in Figure 2.5, which contains the Header field and is followed by an Entity body that contains the data. The Header field contains

additional fields such as requested line, general header, and request and response line, which are used for different request purposes. The most important header request line contains the methods such as GET, HEAD, OPTION, POST, PUT, DELETE and TRACE.

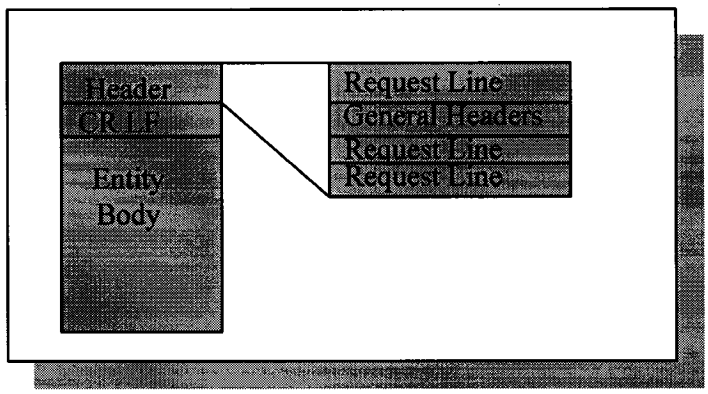

Figure 2.5: HTTP 1.1 request format

GET is one of the most important and common methods used for fetching the web objects from the source. As an example, the "GET/index.html HTTP/1.1" request returns the index.html file to the client.

HTTP Response

The response message, similar to the request message, consists of the header and body as shown in Figure 2.6. In this case, the header contains the status line, general header, response header and entity header. The status line contains the status code of a response such as "HTTP/1.1 200 OK", indicating a successive response. Table 2.1 shows the five different status code categories which are used for web engineering.

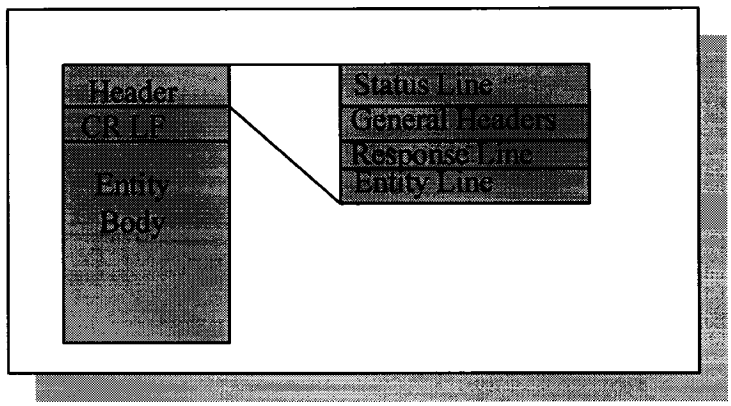

Figure 2.6: HTTP 1.1 response format

Status Code	Response Category
1xx	Informational
2xx	Successive
3xx	Redirection
4xx	Client error
5xx	Server error

Table 2.1: HTTP response status code

2.1.2.3 HTTP and Cache

In the previous section, we have seen the HTTP request and response mechanisms within the client server architecture. The web engineering architecture is a client-server model since that is what HTTP works as. It is very important to understand the different HTTP headers which are related to caching and cache validation purposes such as conditional headers, age and expiration, request redirection, and cache control header. The "conditional headers" are used for caching: "age and expiration" are used to decide when the validity of the cached object is to be checked; request redirection is used to redirect the request; and the cache control header is used for controlling the validation of cached

objects. In the following section, we will discuss some of the most important caching features and the purpose of the HTTP protocol which we use for caching and cache validation. According to the [RFC 261] the following terms are defined as:

Conditional Request:

The conditional request specifies certain conditions in its header field. The server executes the conditional request and responds to the client with a message body only when the condition is satisfied. If the condition is not satisfied, the server responds to the client with either the status codes "304 Not Modified" or "412 Precondition Failed". The following two conditional headers are common for caching;

- **if-modified-since** (last modified date)**:** The purpose of this condition is used to return a copy of a web object that was modified since the specified time.
- **if-none-match** (ETag)**:** Executes the request if the ETag of the requested object if it is different.

Age and Expiration:

Age and Expiration is an important feature of HTTP 1.1. It checks when a cache needs to verify the validity by using the Time to Live (TTL) algorithm.

- **expires** (date)**:** Cached objects will be valid until the expiration date.
- **max-age** (seconds) : maximum age the object may reach before validation
- **age** (seconds): estimated time since the object was served or last validated by the original server.

The Cache-Control Header:

The "**Cache-Control**" header field provides a list of directives that are responsible for controlling the use of caches. The cache-control headers are used for both requests and responses.

Cache- control directives in request:

- **no-cache:** A cached object is never used to satisfy the request. This directive forces to retrieve a new copy of the object from the original server.

- **no-store**: The response from the server is never stored into the cache.
- **max-age** (seconds): Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds
- **min-fresh** (seconds): This header indicates only cached objects that will not expire for a specified time.
- **max-stale** (seconds): This directive indicates that the client will accept a response that has exceeded its expiration time.
- **only-if-cache:** The Proxy server should not forward the request on a cache miss

Cache- control directives in response:

- **no-cache:** Response must not be cached.
- **no-store**: The response cannot be stored into the cache.
- **private**: The response can be reused only by the user who originally requested it.
- **public**: The response is cached as sharable for all users.
- **must**-revalidate: Cache must be revalidated before providing it to users.
- **max-age** (seconds): Cache must be validated before serving the client if the objects reach a specified time.

2.1.2.4 Cache consistence and validation

Cache consistence refers to the integration of the cached data into the cache storage and the accuracy of data between the local cache copy and the original server copy. This is a technique to ensure the cache data is same as the data found on the original server. It is very important for the caching system to be consistent with the cache content since the contents are changing accordingly to their priority [10]. It is useless if the cached copies are not up-to-date with the original server data.

The caching system creates a huge number of copies of web objects over the Internet when clients visit or download web pages through the Proxy cache system. If an object changes in the original server, all the cached copies of the objects are stale. This is a big challenge for the cache system to keep the freshness of all cached copies, called "cache

consistence". Cache consistence is an issue due to the loose coupling between original server and the Proxy cache system.

To remedy this problem we need a special mechanism referred to as a cache consistence mechanism to refresh all cache copies simultaneously to the original server. Otherwise, the Proxy may continue with stale cache copies of the objects.

2.1.2.5 Cache consistence management

There are many different ways to manage cache consistence including time-to-live, client-polling, server-driven, also known as invalidation protocols. Client-polling refers to cache validation and is an approach where the client verifies the validity of the cached objects with the original server. With the invalidation protocol or the server-driven approach, the original server notifies the clients if any objects have been modified at the original servers' end. In the server driven approach, objects are marked as invalid cache when they are notified by the original server; otherwise, the cache is remains valid.

Time-To-Live (TTL): This is a very simple approach that estimates the lifetime of an object and is used to determine how long the cached object remains valid. It assigns a time for TTL for every object and the object is valid in the cache until the assigned time expires. The object will be invalid when the TTL elapses and the next request for that object will be the server from the original source.

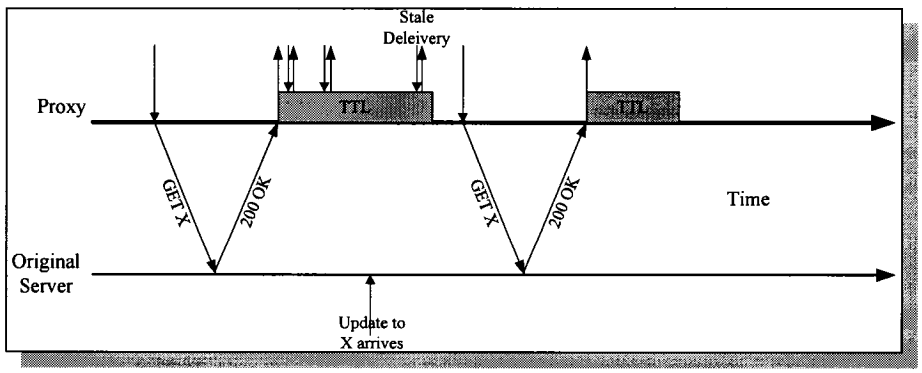

Figure 2.7: TTL-based cache validation and stale delivery

This technique works effectively in various publications including news-papers and magazines.

Validation or Client Polling: The traditional Client Polling protocols present weak consistency because the objects normally check the validity in a periodic manner. The client-polling approach uses HTTP wherein the client runs a query at the original server if the objects are updated. The requested objects are either modified or unmodified. Research [24] showed 20% of requests to the servers could be client polls to revalidate unmodified objects. Thus this validation technique increases the workload of the network and the server as well as read latency.

Client-polling protocols simplify the original server because the protocols do not require an extra workload to maintain cache consistency. In the client polling approach server responses to the client enquire for freshness of a specific object.

Invalidation or Server-driven: In server-driven or invalidation approaches, servers inform the clients immediately after any modifications are made to any objects in the original server proposed by [22]. Usually the invalidation consistence protocol uses a client web trace for both unicast and multicast [36]. The advantages of this approach are that the client has access to a refreshed copy from the original server without any client side validation query. The disadvantage of this approach is to maintaining a list of clients at the server side called invalidation overheads. Another disadvantage of this system is the requirement to send invalidation messages to the client even if the cache object is no longer in cache producing overhead at the server side as well as at the client side.

Leases: The lease proposed by Gray and Cheriton [19] is a mechanism to improve the server-driven approach. A lease is assurance of notification by the server to the client whether any object was modified during the lease period. The server must keep a copy of the client list until the client lease expires. Figure 2.8 shows the basic mechanism of a lease:

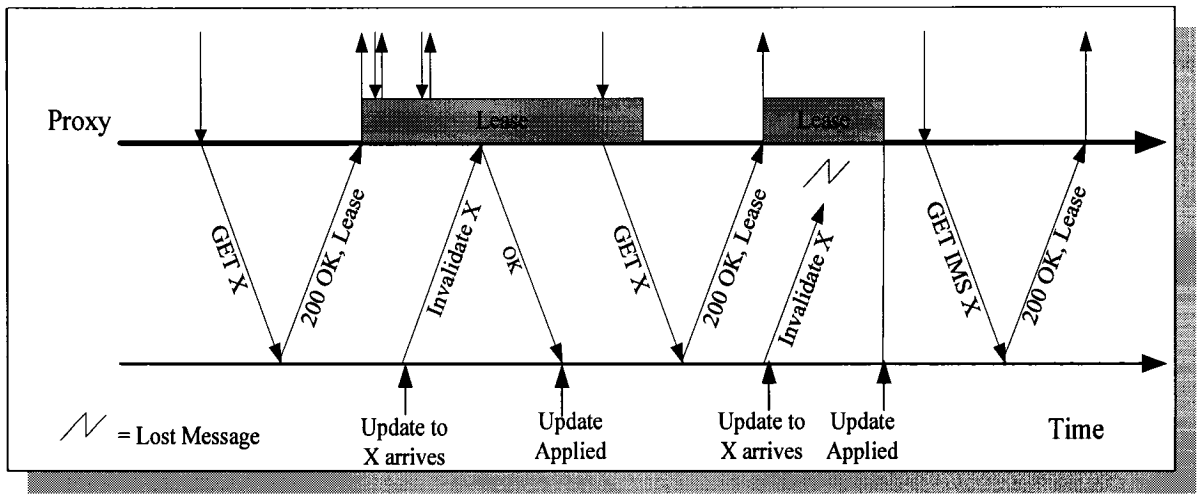

Figure 2.8: Cache invalidation with lease

2.1.2.6 Cacheability

Cacheability of web content means web objects can be cached depending on some important parameters or criteria. Normally the objects are cacheable if:

- the objects have long freshness time;
- small size objects;
- high access latency;

Objects are uncacheable if:

- objects are password protected;
- there are any URLs with /cgi-bin/ or any pre-configured patterns (such as "?");
- dynamically generated responses from a query result;
- request and response with cookie and set cookie header;
- any files exceed the pre-defined limit;
- SSL requests are tunneled through and not cached

2.1.3 Cache Replacement

The Proxy server faces several policy decisions as an example of object's evection from cache storage, also known as cache replacement strategy. The web object evections are required when there no more space is available to store new or additional web objects in to the cache storage. Many popular replacement strategies are researched and

implemented for cache replacement such as LRU (Least Recently Use), LFU (Least Frequently Used), Greedy-Dual-Size, Greedy-Dual*Web as shown in the Table 2.2. Most of the algorithms are based on the web object's usage parameters such as access recency and frequency of object accessed, size and type of the objects, the cost of downloading particular objects and cache space size etc. Aggarwal et al. [3] categorizes the algorithms into three different groups of replacement algorithms. In his first category, he considered a classic replacement algorithm, wherein the LRU replaces the web objects that were used least recently and LFU replaces the objects that were used least frequently.

Algorithm	Definition of eviction	Recency of accessed objects	Frequency of accessed objects	Age of the object	Size of the object	Cost to request particular objects	Cache space size	User web behavi our
LRU (Least Recently Use)	Least recently first	X						
LRU-SIZE (Threshold)	Remove largest size	X			X			
LRU-MIN	minimize the number of documents	X						
PSS (Pyramidal Selection Scheme)	Removed the largest value of SiDeltaTi	X						
LFU (Least Frequently Used)	remove the least frequently requested objects		X					
LFU-Aging Arlitt et al. 2000 [6]	uses a maximal value for the frequency		X	X				
SIZE	Evict largest object first				X			
Greedy-Dual-Size			X		X			

Table 2.2: Cache replacement algorithms and associated parameters

The second categories are based on the primary key (the attributes) of an object such as the size of the objects. Williams et al. [44] found in his studies that the best hit rate occurred when he used a simple key that evicted the largest objects first.

Another group of algorithms is based on the combination of classic and key-based algorithms such as LRU-threshold. These threshold values are dependant on different keys such as the size (small and large) of the objects, latency cost, and so on. This group of algorithms is not only based on the primary key on size but also depends on the other parameters such as the cost of objects. The cost-based algorithms are based on the different factors such as time last used, usage frequency, and cache transfer time, time of day, object's HTTP header and many more. Cao and Irani derived a popular Greedy-Dual-Size algorithm. In this algorithm they considered the cost of each object and evict the object from the cache with lowest cost/size ratio. Jin and Bestavros [26] showed another algorithm the Greedy-Dual* Web caching replacement algorithm, which based on relationships between objects. In this approach they consider the relationship between objects because the objects belong to the same document: if retrieved together, they will most likely be retrieved together again.

2.1.4 Web Prefetching

Prefetching is another well-known technique for reducing user web latency by preloading the web object that is not requested yet by the user. In other words prefetching is a technique that downloads the probabilistic pages that are not requested by the user but that could be requested soon by the same user. As established there is some elapse time between two consecutive requests by the same user. Prefetching usually performs the preloading operation within an elapse time and puts web objects into the local browser or Proxy cache server to satisfy the next user's requests from its local cache.

Padmanabhan and Mogul [33] in their research show successful prefetching of web objects into local caches, which are used to further reduce web response times. The main idea of prefetching is a prediction algorithm or technique is used for selecting the web objects that are not yet requested by the user but that could be soon. The main function of the prediction algorithm is to select the web objects by analyzing the user's previous web actions. The user's web actions can be determined by the web access log, user browser navigational activity, browser tools and the user's related information. The user-related information includes the age, sex, and professional and personal web interests of the user.

The user-related information is very useful for determining the user's web behaviour because the user's behaviours depend on his or her age, sex and interest. The user-related information can be determined by collecting information in many different ways including user interviews and explicit feed back or monitoring the user's web behaviour studied by [32], [12].

The web browser makes a request for the web object on behalf of the human. The internet requests are the user's web interests or behaviour on the Internet. User web interest is a key factor for prefetching techniques because users are always interested in visiting pages with topics of interest even those are new or have not been visited before. A successful prefetch prediction depends on the human Internet behaviour. Determine human web interest or behaviour is very complex since their interests change over time. The behaviour of a user changes due to the changes in their occupation or age. In the next section, we will see the relationship between human behaviour and the Internet interests.

2.2 Web Behaviour and Neural Network

2.2.1 Human Web Behaviour

Many prediction algorithms are based on the past history of a user's web action or behaviour. Some prediction algorithms explicitly consider the information history of user web access. The user's web behaviour is determined by the user's web access pattern and web navigational activity. We will discuss in details human web access behaviour in the following sections. User behaviour on the web is a relatively new and complex area of study. Users' web interests are not static and therefore change over time. Web interests depend on the user's sex, age, occupation and current lifestyle including, position, time and location.

2.2.1.1 Measure human behaviour

Initially, a user does not read the whole content of a document. He or she just scans the document, particularly the links. During the scanning, the user makes a decision within a few seconds; he or she will leave the page, stay or explore a new link from a current page

depending on the topic of the page. Users leave the page and can go to either a new page or back to the previous page or can exit from the current page. Users can go to new pages by clicking the link from the current page or by typing the link to the address area in the browser if he or she has found his or her interesting topics. In general people use the first method to select or browse the pages of interesting. A user normally selects a new page or link by reading the anchor text. The user also selects the pages or links through browser tools such as history, favorites and bookmarks etc.; however the clicking action is most popular.

There are a number of ways to measure human web behaviour, but the most popular strategy is the web usage pattern. The pattern of web usage can be determined by recording the user's click-stream actions (logging) while on the web. There are some advantages of this click-stream logging; it can be recorded from any of the sides, end users [34], through a Proxy server [21] or through a server end [4]. Other advantages of this approach require no external interaction for determining this strategy.

2.2.1.2 Changes in human web interests

The users' web interest or behaviour changes with time; today's interest could not be tomorrow's interest. A graduate students' current interest might only be his or her research field but that could change soon after his or her graduation. A professional changes his or her web behaviour depending on how his or her field of occupation or expertise changes with time.

User behaviour or interest can be categorized into two parts: static and dynamic. Dynamic interest changes very frequently depending on the current situation/state but static interest is a long run interest that does not change frequently. After a certain period of time, some dynamic behaviour changes to static behaviour, continuing for a long period of time.

2.2.1.3 Tracing human web interest

The user's web behaviour can be studied through the user's browser actions; already discussed in section 2.2.1.1. This actionable view of user behaviour can be monitored by the following actions:

- A clicking action by the user;
- The sequence of clicking action;
- How the user behaves or reacts;
- Time spent on a page or session;
- Content they are browsing

2.2.2 Personalized Web Information System

In this section we will discuss the Personalized Web Information System and how it is useful for caching and prefetching. As we discussed earlier in this chapter, web interest is a key parameter of the personalized caching and prefetching system. It is possible to build a personalized web information system (user profile of web) by analyzing the user's web interests. For developing a web personalization system we need to develop a details user profile of user web interests. The user profile can be built by the user history of the web usage pattern or by collection of the user web related information through interview process.

The personalization information system is mainly used for predicting web navigation and assisting the development of the personalization of web information including personalized content and results. With the help of the personalized web information system, the prefetching system anticipates future requests or provides guidance to the client. If a personalized prefetching system correctly predicts the next page and preloads those that would be visited the user latency of the next request would be greatly reduced. By using the personalized web information system, a personalized web caching system can also be constructed.

2.2.2.1 Personalized caching and prefetching system:

Personalization of "Caching and Prefetching" system provides a caching system based on his or her web interests. The personalized caching and prefetching system is very useful for the mobile user who frequently moves from one network to another. The advantages of this system are:

- Cached only user relevant objects
- Privacy protection since the cached item is not shareable with other users
- Cache moves with mobile devices
- No need higher for range of hardware for personalized caching or prefetching due to lower workload

2.2.3 Neural Network:

In the early 1940s McCulloch, a neurophysiologist, and Pitts, a mathematician introduced neural network artificial intelligence in the computing field. Neural network algorithms came to the practical field around late 50s for non-traditional problem solving. Some modern popular neural network techniques are used in research, including Hopfield Nets, Competitive Learning Models, multilayer networks, Boltzmann machines and adaptive resonance theory models [23]. Many researchers are using this artificial intelligence to solve problems where traditional solving algorithms are not suitable. Hassoun [20] proposed that when there are no good algorithmic solutions available for problems, use neural network. In the next section, we will see the Neural Network components and techniques.

2.2.3.1 Neurons

The fundamental processing unit of a neural network is a neuron. A neuron receives inputs from different sources and then combines them in a fashion to perform a nonlinear operation on the result and produce an output as a final result. The basic unit of neural network is shown in Figure 2.9.

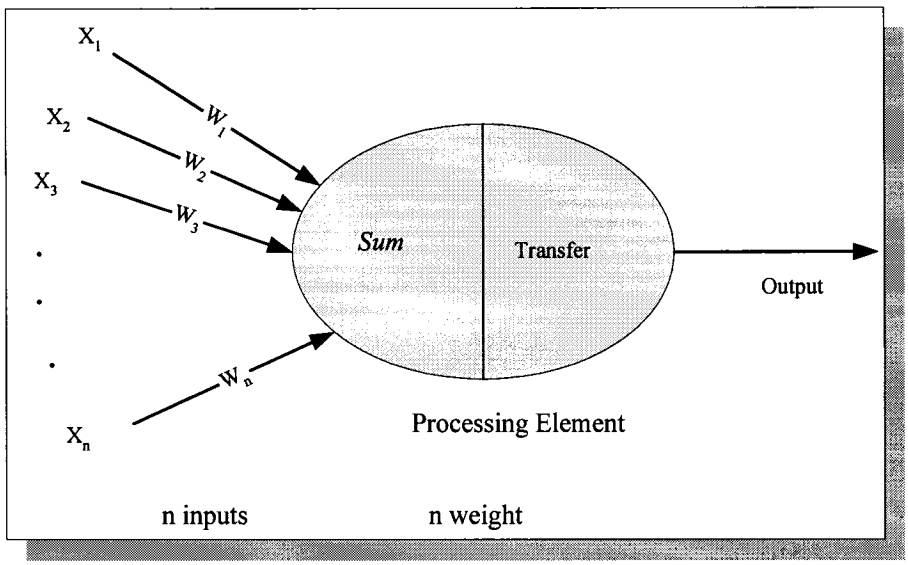

Figure 2.9: Basic diagram of a simple neural network

As demonstrated in the above Figure, the inputs X_1, X_2, X_3 .... X_n are connected with their respective weights W_1, W_2, W_3, ...W_n. The products of input and weight are simply summed, fed through a transfer function to generate a result, and then finally produce an output. The transfer function scales or controls the output value via a threshold value. The most common and popular transfer functions are sigmoid, sine and hyperbolic tangent.

Figure 2.9 is a very simple one-layer network diagram. For improved performance, the researcher used multiple hidden-layered networks. Generally, different layers are defined as input, hidden, and output as shown in Figure 2.10. The hidden layer could contain several hidden layers.

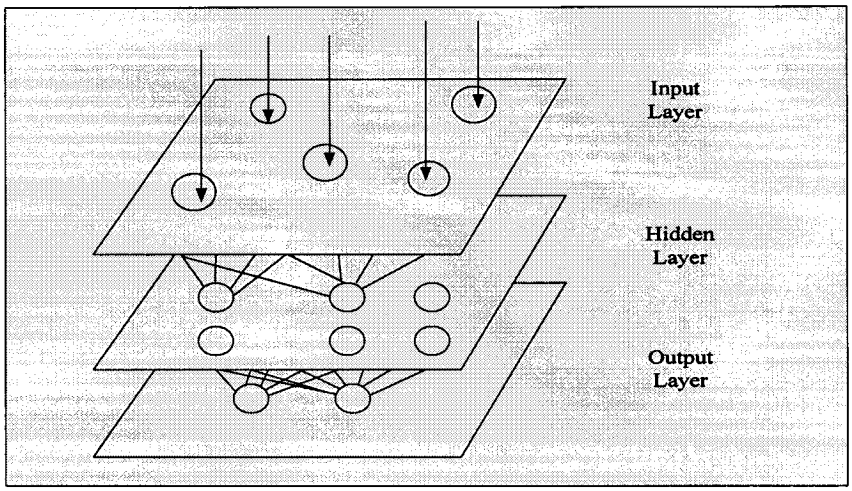

Figure 2.10: Simple multilayer neural networks diagram.

The hidden layer neuron receives the input from all other neurons from the above layer and passes its processing output to all of the neurons below this layer. The lines between the neurons establish a communication channel which is essential for network training.

2.2.3.2 Multilayer Perceptron (MLP) with back propagation

A typical back-propagation network has at least three layers an input layer, an output layer, and at minimum one hidden layer. The hidden layer could contain several layers. There is no theoretical limitation to the number of hidden layers. Typically two or three hidden layers are used to solve problems of any complexity.

Both the networks' input and output layers indicate the flow of information during recall. The recall of neural network back-propagation systems is the process of inputting data into a trained network and receiving the result. Back-propagation techniques are used for network learning time and are not used during recall [11].

In a multilayer back-propagation neural network the final output (recall) result is compared to a predefined threshold value; if the result is less than the threshold value,

then the neuron yields an output of zero. Otherwise, it results an output of one for all cases, as shown in the equation 2.1.

Consider the network below that contains one hidden layer that consists of hidden neurons and one output neuron. For a current set of weights if an input vector is propagated through the network, then an output d(n) will be produced. This output d(n) value could not be the same as the required output y(n). The difference between these two outputs is referred to as an error. An algorithm is required to reduce the errors. The back-propagation is a popular algorithm to reduce such errors. The back-propagation algorithm produces a weights value, which minimizes the network errors. The back-propagation is an algorithm that performs a gradient descent minimization of the network's total mean squared error, e^2.

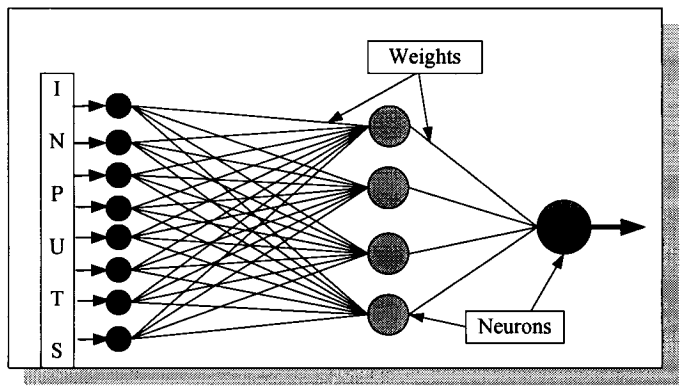

Figure 2.11: Single hidden layer neural network weight

The network output is:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \\ 0 & \text{Otherwise} \end{cases} \qquad (2.1)$$

The network error of outputting a neuron, j, after the activation of the network on the n^{th} training set $(x(n), d(n))$ is:

$$e_j(n) = d_j(n) - y_j(n) \tag{2.2}$$

The network error is the sum of the squared errors of the output neurons:

$$e(n) = \frac{1}{2} \sum_{j \text{ output node}} e_j^2(n) \tag{2.3}$$

The total mean squared error is the average of the network errors over the training examples:

$$e_{AV} = \frac{1}{N} \sum_{n=1}^{N} e(n) \tag{2.4}$$

The back-propagation weight update rule is based on the gradient descent method: take a step in the direction yielding the maximum decrease of the network error *e*. This direction is the opposite of the gradient of *e*.

2.2.3.3 A generalized back propagation algorithm

```
N=1;
Initialize weight W (N) randomly;
while (stopping criterion not satisfied and n <max_iterations)
   for each data set example (X,D)
      run the network with input X and compute the output Y
      update the weights in backward order starting from those of the output layer:
      with compute using the (generalized) Delta rule
   end-for
 N = N+1;
 end-while;
```

2.3 Related Work to Caching and Prefetching

Most of the prediction algorithms are based on information accessed by past users. There are some popular prediction algorithms in existence in the industry and academy. Most of the existing popular algorithms [15], [38], [39] are based on the history of the user's access pattern. History-based algorithms depend on the URL accessed pattern. The accessed patterns are determined by the accessed log file. This means history-based prediction works only for already visited URLs or pages but does not work for new pages that are not yet visited. Content-based prediction is a popular prediction algorithm, which is based on the users' preferred words referred to as keyword in the anchor-text of links. In the next section, we will see some popular prediction algorithms.

2.3.1 The Top 10

Markatos and Chronaki [29] proposed the "Top 10" prediction strategy which is based on the popularity of the web document. Top 10 approaches are based on server and client cooperation which use server active knowledge of the most popular documents (their top ten) with the client access profile [28]. Periodically, the server builds a list of top web documents by using its ten most accessed documents. Any of the documents within this list are considered prefetchable documents for clients. Markatos and Chronaki claims top-10 approach can anticipate more than 40% of a client's requests and traffic does not increase by more than 10%.

2.3.2 Markov Modeling

The Markov model is sequenced - an ordered set of user web action-based prediction model. The basis of this model, a prediction of the next request, is a function of the current state, with each state representing the sequence. A typical Markov model, often called the Markov graph or tree, containing the web objects that represents the nodes, and directed edges represent the weight between the two states shown in Figure 2.12. The weights are the probability of the corresponding objects transition. The main basis of this

model depends on the number of nodes and the possible paths. The prefetch prediction would improve if more nodes and paths are used.

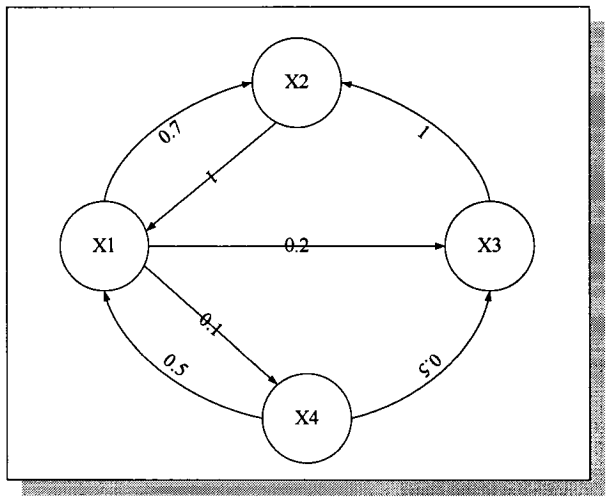

Figure 2.12: Markov Graph

Different Markov-based algorithms use different notations of object transition when building a Markov graph. Nicholson and Zukerman use the classic model of the Markov model wherein the transition occurs from A to B if the same client accesses B immediately after A. The study of Padmanabhan and Mogul [38] considers that a transition occurs from A to B when the same client accesses B within a certain number of object access after and Bestavros [9] uses certain time T. A general thought of the Markov tree algorithm is a larger tree produce a high performance prediction. Researchers use different order of the Markov tree such as Zukerman, Padmanabhan [38], Michael Zhen Zhang and Qiang Yang [40] and many other researchers have used the first-order Markov tree as well as some who use 2^{nd} order. Themistoklis Palpanas and Alberto Mendelzon [41] used the 4^{th} order Markov tree model for a better prediction. A higher order tree produces better results but overhead costs more than lower order. Different research [41] and Zukerman shows that Markov tree prediction approach can anticipate more than fifty percent of clients' requests and traffic does not increase by more than twenty percent.

There are some noticeable disadvantages of the Markov large graph model such as higher cost, computational time, and storage.

2.3.3 Prediction by Partial Matching (PPM)

Prediction by Partial Model is another kind of prediction approach to incorporate the context into the prediction mode. Fan [42] proposed and implemented this prediction algorithm which is based on the Markov model. It is a higher degree of the Markov model that is based on three basic parameters: the number of past requests, the number of future requests predicted p, and the probability threshold value. To find out the prediction candidate, the algorithm determines all instances in the history structure that matches any suffix of the client last m request. If the probability of a URL exceeds the threshold value P, then the URL is selected for prefetching. Fan [42] shows in his simulation result that the latency is reduced by 17-23.4%.

2.3.4 Artificial Neural Network Prediction

There only a few numbers of neural network-based prediction algorithms are available. Researcher Wen Tian, Ben Choi, and Vir V. Phoha [35], Tamer I. Ibrahim and Cheng-Zhong Xu [37] proposed neural network based predictive prefetching algorithms. In this section we discussed their system and predominance.

Wen Tian [35] proposed an approach that uses the back-propagation learning rule in neural a network module. The predictor module contains two modules: one is the preprocessing module that is responsible for filtering and preparing data in a format that allows it to be fed to the processor and the second module is a processing module which is responsible for training the neural network by using the data obtained from preprocessing shown in Figure 2.13.

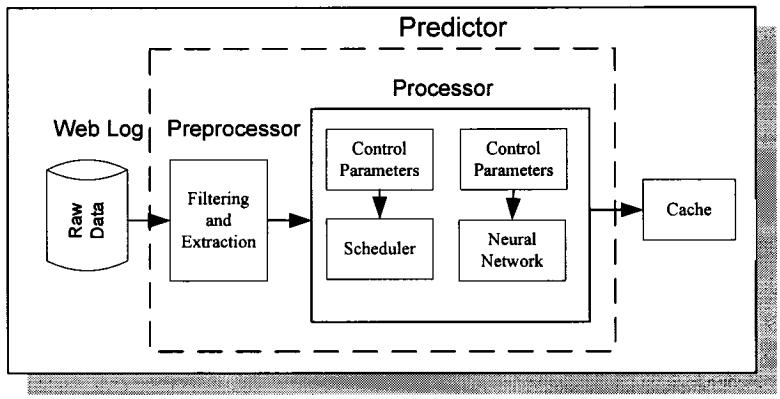

Figure 2.13: Schematic diagram of a neural network-based intelligence prediction

This prediction technique is based on the history of web accesses to a web site. In the net training session, different windows, such as training window WT, the size of NT, backward-looking window WB; and a forward-looking window WF are used for the normalization of their input value. They claim that the accuracy of prediction is up to eighty-six percent.

Tamer I. Ibrahim and Cheng-Zhong Xu [37] proposed prediction algorithms that are based on keywords in anchor texts of URLs. The predictions are made by neural networks over the keyword. The prediction unit extracts the keywords from the URL and adds new words to the keyword lists: in other words, the prediction unit increments the weight of the words. The preferences are computed with the associated weight of the keyword and then compared to the neuron threshold value. If the computed value is higher than the threshold value it is then placed (URL) into the prefetch queue. If the prefetch queue is empty, it will start to process the next links. This research claims the experimental results achieved approximately sixty percent hit ratio and less than thirty percent undesired network traffic.

Chapter 3: System Design

3.1 PROTEUS ARCHITECTURE

In this chapter we will present the design and architecture of Personalize and Artificial Intelligence Proteus. Preliminary, we will illustrate the architecture of Proteus, and then we will present the Proteus software design technique with the aid of UML diagrams including package diagram, class diagrams and sequence diagrams.

We implemented a "Personalize and Artificial Intelligence Web Caching and Prefetching" system by extending the existing Proteus system. A basic architecture of the Proteus is shown in Figure 3.1 which consists of a Proxy module and simulation module. Proteus performs two major tasks: one for real-time Proxy that includes caching and prefetching and the second for simulation of the request generator. If Proteus operates in Proxy mode, in the real time operation, the system consists exclusively of the Proxy module and if the Proxy is used for simulation, the "Request Generator" generates the requests that run as a simulator. We discuss only the caching and prefetching module in this study the main focal point of this study.

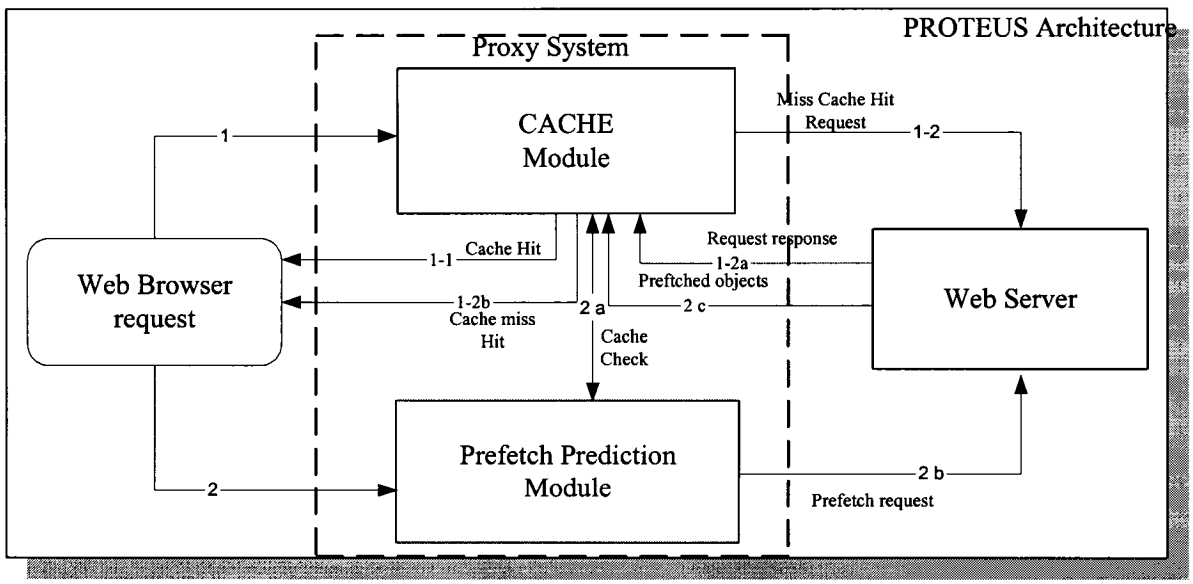

Figure 3.1: Basic architecture of artificial intelligence Proteus

We will discuss the individual details of the design diagram of caching and prefetching in the next few sections.

3.1.1 Proxy Module

The Proxy module is the main module of the Proteus that delegates the user's request and Internet response to the client and, as well, provides web caching and prefetching functionality. It intercepts all users' requests that are sent to the server to verify if the Proxy can fulfill the request itself. If the Proxy is unable to satisfy the user request, it forwards the request to the original server. There is no direct communication established between the client and the server but both are connected through the Proxy module as show in the Figure 3.1. Proxy performs not only as a delegation of the user's requests but it also provides some additional functionality such as storing the web objects into the cache during the server's response to the client and prefetching the web object before the client makes a request. Proxy performs some additional activities such as filtering Internet sites, filtering objects that are stored in the cache (cache removal) and traffic communication. In the following sections, we will discuss the design and functionality of two major components of caching and prefetching and their operational techniques.

3.1.1.1 Cache main module

The cache main module plays the main role for storing and managing of web objects into the cache storage. The Figure 3.2 is basic architecture of a cache main module which consists of two sub modules: cache module and cache validation module. The cache module stores the web objects in its local storage once it is satisfied with the cache strategy. It serves the web objects to the client once the client request is satisfied by the local cache. It generates errors when it fails to successfully download the requested objects from the original server due to the dead Internet links, traffic problems, application problems etc. This module is also responsible for selecting the caching strategy and cache filtering which we will discuss in the next section.

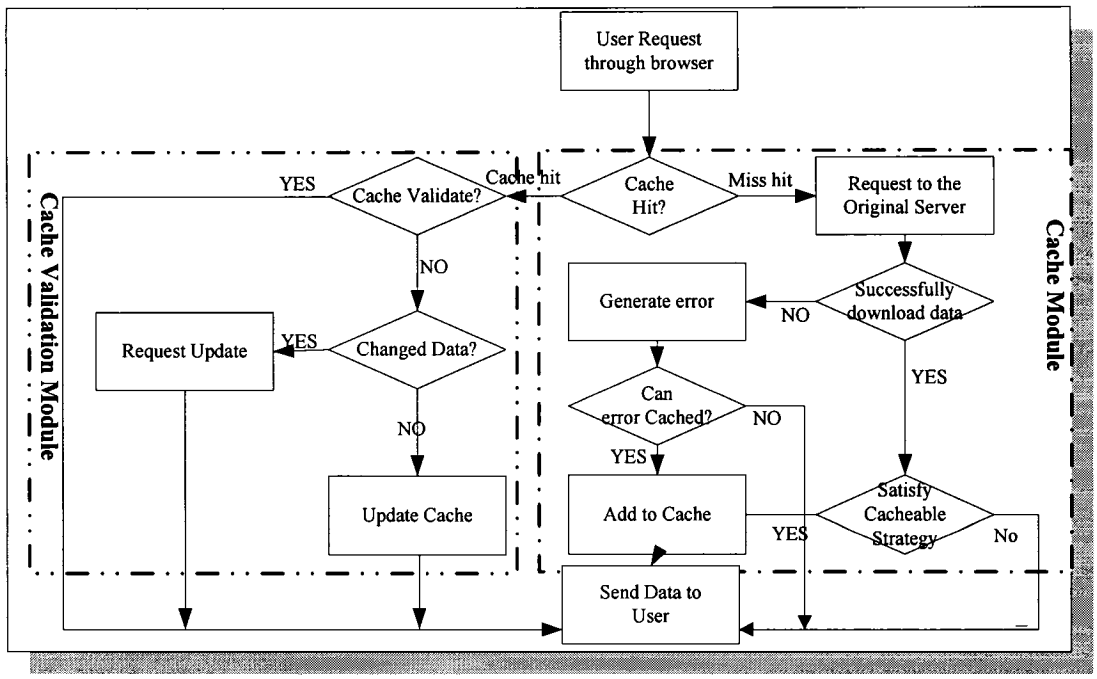

Figure 3.2: Basic design architecture of a cache module

The cache main module system generates a certain shelf life called TTL for objects. This is a time stamp used for cache validation. While the objects are fresh and requested by the user, they will be served directly from the cache. If the objects' TTL is expired, the cache validation module checks with the original server that the object is the same or has changed. If it has changed, then it will request a new object or update the object to the original server as shown in Figure 3.2.

3.1.1.2 Cache initialization process

The cache initialization process starts once the Proxy starts. During the initialization, it performs a series of operations shown in Figure 3.3. The "create cache" process sets an environment for the cache system. The main tasks of this process are processing and defining the cache-related parameters:

- the name of the cache listing where the object can be stored
- the name of the configuration file where the configuration is to be stored
- the name of the cache database

- the name of the "don't cache vector" database
- the name of the log file, where all events of the cache module are logged,
- the name of the caching strategy which can be initialized during cache initialization
- the maximum size of the caches in bytes

During the time of cache initialization processing, the cache module calculates the number of objects in the cache and number of accesses to the cache and also determines the characteristics of the cache objects. In the next step, the cache module also verifies whether the object exists or not. If the object exists, the system will compute priority for that object by using the current strategy or algorithm. If the priority is higher, the threshold value and the files are placed into the cache and the cache status is updated. If the files' priority is lower than the threshold the file is removed from the cache.

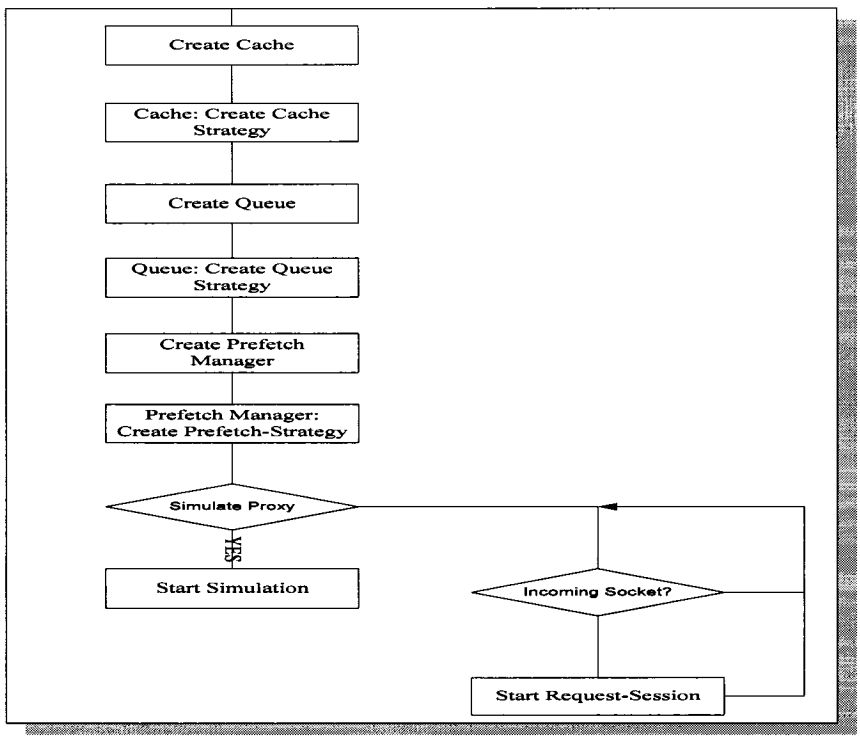

Figure 3.3: Cache initialization process

3.1.1.3 Cache strategy module

Cache strategy is one of the most important modules for cache management that starts during the cache initialization process. Cache strategy makes the decision of what object would be cached and what would not be cached or evicted. If the assigned space of the cache is fully occupied and a new object needs to be cached and/or updated, the cache strategy module allocates the required space by deleting low priority objects. The cache strategy module first determines which files are to be deleted from the cache by computing the object's priority. The object's priority is determined by analyzing its properties. In the next step, the cache strategy module will compute how much space needs to be deleted from the cache.

Figure 3.4 shows the sequence diagram of the cache strategy that starts if the objects are not rejected by the "don't cache vector". The "don't cache vector" consists of a database that contains the names of files or file extensions that should not be cached.

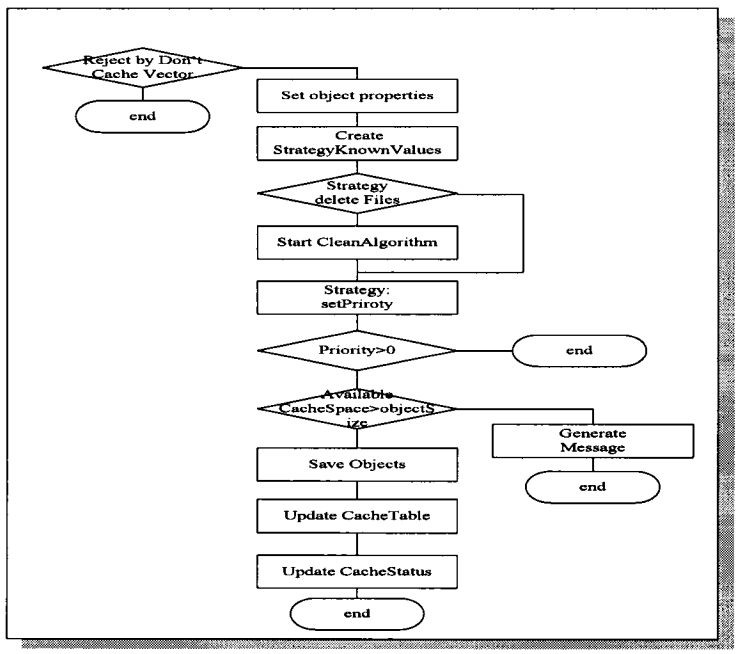

Figure 3.4: Cache strategy for a new object

Proteus is designed in a flexible way so that it can easily implement many different strategies. The strategies are merged into the main Proteus module during the initialization of the respective modules. Thus, it is possible to select a new strategy during the Proteus run time without changing the program code.

Figure 3.5 depicts our proposed artificial intelligence cache strategy system architecture. This strategy is based on the neural network. The neurons of the neural networks are users' web behaviour, and different web objects' properties such as object type, size, age, retrieval time, etc. Individual modules of the cache strategy system will be discussed in detail in the prefetching sections. In this module, we designed the neural network with the same inputs as prefetch except that the input values are calculated differently.

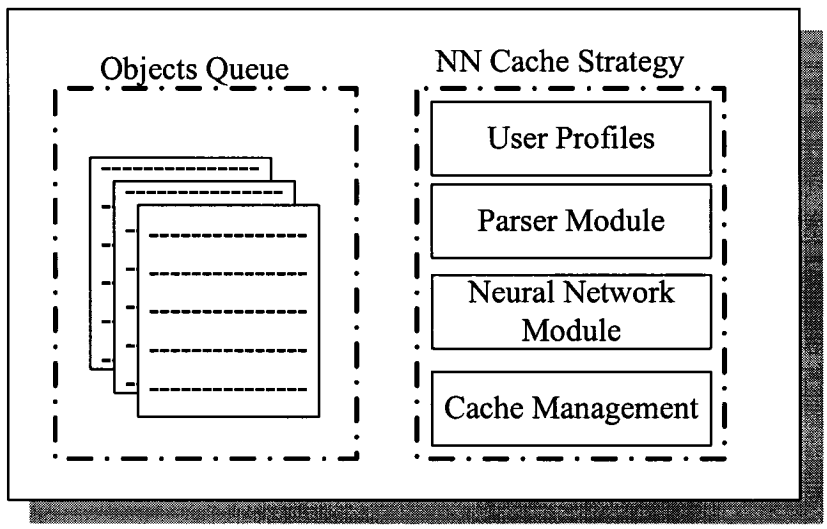

Figure 3.5: Basic architecture of cache strategy module

1. The "Object Queue" module queues all the downloaded objects for caching. The "Object Queue" modules push the queue to the "NN Cache strategy module" for caching.
2. The "Parser" module parses the "Tile of the page" and extracts the keywords and calculates the keyword ranking with static and dynamic profile. The parser module also computes the associated object properties such as object's URL, size, type, age, and retrieval time.

3. The "Neural Network Module" computes and sets the object's priority by using the multilayer neural network and then compares it to the priority threshold value. If the object's computed priority is higher than the priority threshold value then the object is marked for caching.
4. The "Cache management" process stores the selected object to the cache and updates the cache database.

3.1.1.4 Artificial intelligence cache removal module

In our research, we found that a number of cache algorithms are implemented by the researchers. In this research, we have proposed a new personalized cache replacement algorithm based on artificial intelligence. In this proposed algorithm, we emphasize the user's personal web interest and other object's properties, such as object size, object type, object age, and object retrieval time. The personal web interests are based on the user's web action, which was discussed in section 2.2.1. The Artificial Neural Network (ANN) cache replacement algorithm is developed based on the user's dynamic and static web interests. The basic design diagram of the Artificial Neural Network (ANN) cache replacement algorithm is shown in Figure 3.6.

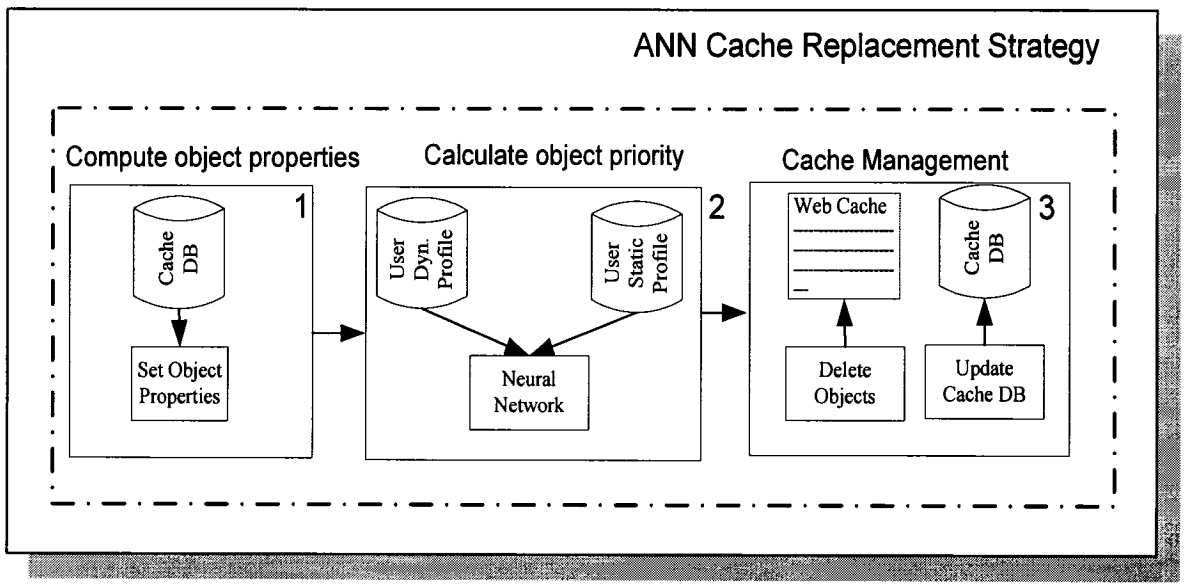

Figure 3.6: Artificial intelligence cache replacement algorithme

1. The "compute object property" module examines and computes the object's properties, such as the object's URL, size, type, age, and associated keywords from the web cache database.
2. The "calculate objects priority" module computes and sets the object's priority by using the multilayer neural network and then compares it to the priority threshold value. If the object's computed priority is less than the priority threshold value the object is marked for deletion.
3. The "cache management" process deletes the selected object from the cache and updates the cache database.

3.1.1.5 Cache Consistency

Maintaining the consistency of the cache is a significant challenge, since files are added, deleted and modified and changed frequently on the server side. The web objects are also frequently modified on the server side, creating a major impact on the caching consistency. The cache consistency module issues "if-modified-since" and/or

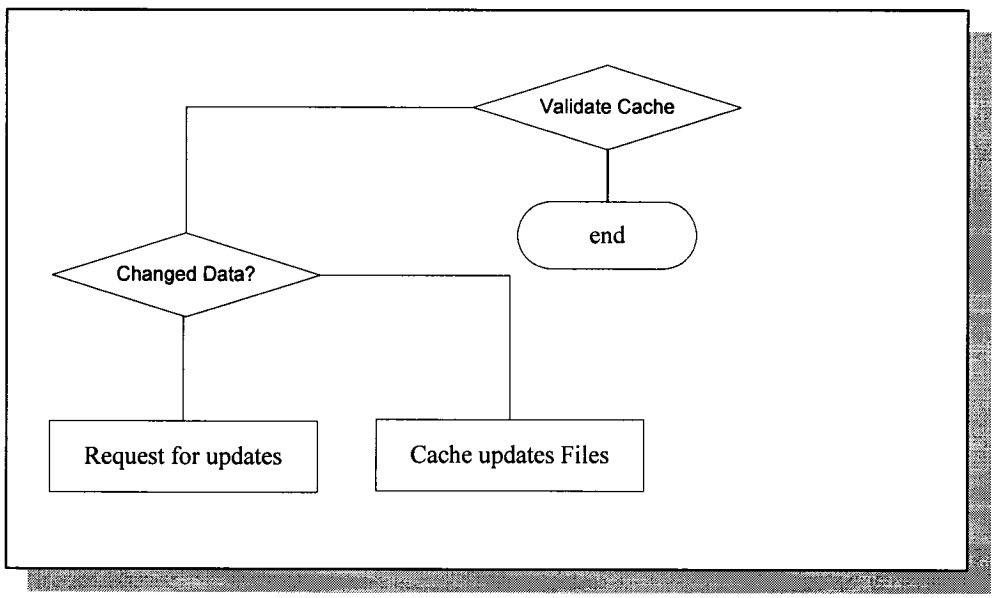

Figure 3.7: Cache consistence management

"cache control" requests to the original server to check if the objects are still valid. This module checks every n day (TTL) if the object is updated or not.

3.1.2 Prefetching Module

The prefetching module regulates the pre-downloading activity of web objects before the user requests them. This module intelligencely predicts which link should be preloaded by assessing the user's previous web behaviours or interests. The prediction algorithm represents the main part of this module. In the following section, the prediction module will be described in details. Figure 3.8 illustrates the basic schematic diagram of the prefetching algorithm. There are few algorithms already implemented in the previous version, Proteus, therefore we are implementing a new algorithm that is based on the artificial intelligence multilayer neural network. The Prefetch Manager is responsible for managing these strategies. The Prefetch Manager and prefetch strategy initialize at the beginning of the proxy server's run time and continue until the Proxy server is stopped.

3.1.2.1 Prefetch system architecture

In this section, the architecture of the artificial intelligence neural network prediction will be discussed based on the prefetching algorithm. Figure 3.8 shows the system architecture of this module, which consists of five sub modules: the keyword parser, the keyword ranker, the neural network, the predictor, and the user's dynamic and static profiles.

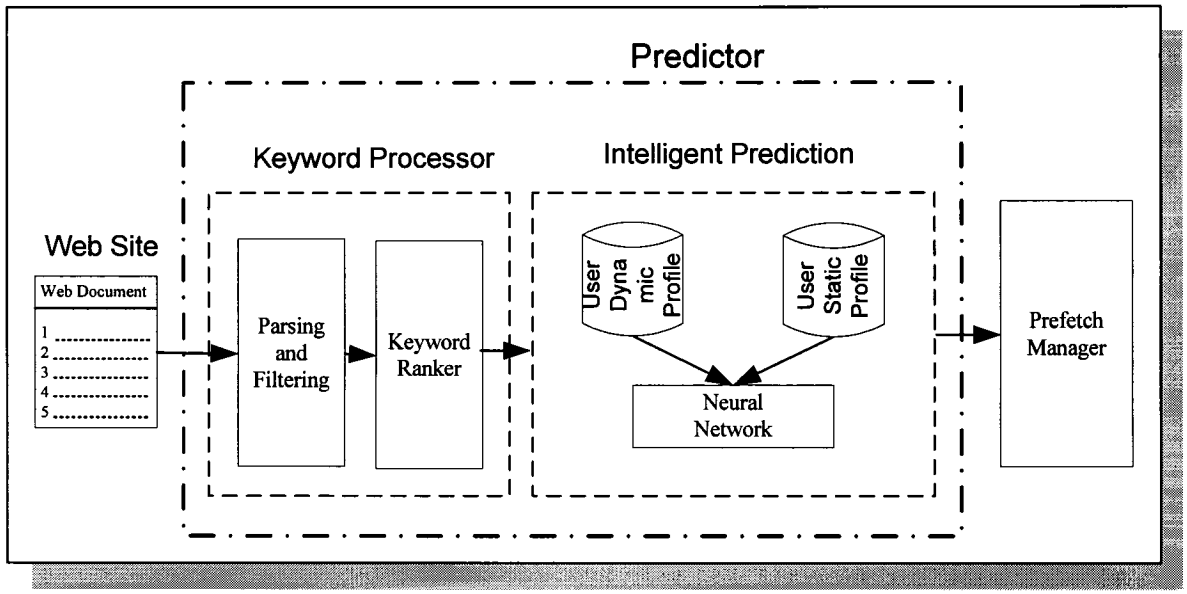

Figure 3.8: System architecture of the AIMNN prefetching system

The intelligence prediction system follows the following these steps to perform the prefetching of a link:

1. The "keyword parser" examines the HTML page and identifies the links. Every link is processed to extract the link associated words until a further user request or click action. This module filters the unwanted words from the extracted words and stored those keywords in a temporary hash table for further processing.
2. The "word ranker" determines the properties such as the size, age, and type of the object associated with the keywords (link). It then calculates the rank of the keyword by using these object properties for the neural network. Finally, the ranking module feeds these keyword parameters into the prediction module.
3. The "prediction module" determines if the keyword associated with the link is prefetchable or not. The neural network calculates the priority of the keyword by using the word's parameters, such as the corresponding value of the user's dynamic and static profiles and other associated parameters that are determined by the previous module, such as age, size, type and learned weight from the neural network learning process.

4. The predictor module computes the priority value of the keyword and feeds it to the prefetch manager. The prefetch manager compares this value with the provided threshold value and decides if the keyword (associated link) is prefetchable or not. If the prediction output value is greater than the threshold value then the prediction module will preload the link associated with this keyword.
5. Steps 2 to 4 are repeated for the next keyword. If the link associated with the current keyword is already preloaded or cached then the process continues with the following keywords.

3.1.2.2 Keyword parser module

The keyword parser module extracts the words from the anchor text of a link and puts them into a keyword list which includes the associated links and their size, type and age. These extracted keywords' parameters are converted to the keyword rank by using a predefined value. In a single link there could be one or more keyword are available. In this situation, the keywords are selected randomly from the keyword list and then tested to see if the keyword is prefetchable or not. If the keyword is prefetchable there is no need to test for further keywords for this current link. This process continues for the next link, and the next, and so on, until a new request is issued or a click action is executed by the user.

Figure 3.9 shows the data flow diagram of the keyword parser module. This module follows the following sequences to parse the keywords:

1. The "link parser" parses the links from the HTML page.
2. The links are checked to see whether they are already in the cache. If the links are not in the cache, then move to the next step; otherwise, start from step 2 for the next link.
3. The "keyword extractor" extracts the words from the anchor text for that URL.

4. The keywords are filtered from the extracted words by using the keyword strategy and the "don't keyword" in order to make a keyword list.

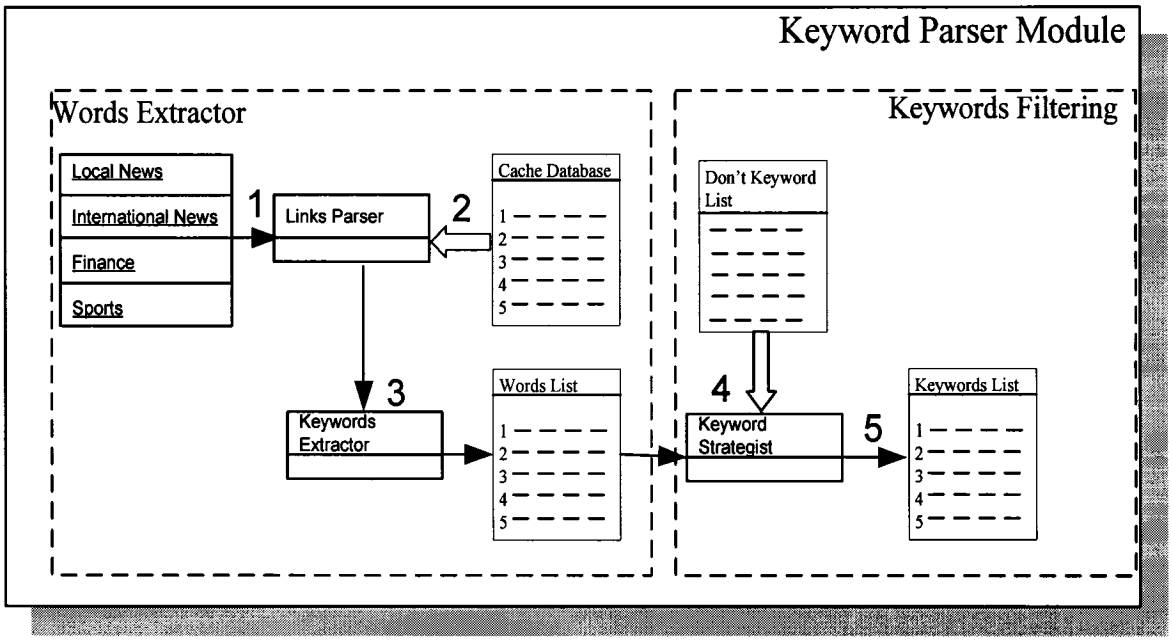

Figure 3.9: Data flow diagram of a keyword parser module

1. The keywords are put into a list (temporary table) with the link, and the other associated parameters such as size, type, age etc.

Keywords: It is very important to filter the keywords from the other words, since the keywords represent the user's areas of interest on the web. In the keyword selecting strategy, we assume the words should be a keyword if they:

1. consist of more than three letters
2. start with a capital letter, and/or
3. are not an HTML tag (remove any HTML tag code if it is in the word lists)

The "don't keyword list" contains a number of words that are:

1. not the user's interest words

2. a list of articles
3. prepositions
4. the verb "to be"/ and/or
5. pronouns, etc.

As shown in the following box (a Google search result), two links were found that contain the text associated with these links. In the second box we can see the HTML source code for the links.

Caching and **Prefetching** for Web Content Distribution (ResearchIndex)
Predictive **Caching** and **Prefetching** of Query Results in Search Engines

 Caching and Prefetching for Web Content Distribution (ResearchIndex)

 Predictive Caching and Prefetching of Query Results in Search Engines</p>

The first link contains the text " Caching and Prefetching for Web Content Distribution" and the second link contains the text "Predictive Caching and Prefetching of Query Results in Search Engines". We can get a keyword lists from the first link after filtering the HTML tags, grammatical words, etc:

1. Caching;
2. Prefetching;
3. Content; and
4. Distribution.

and we can get the following keyword from the second link:

1. Predictive;
2. Caching;
3. Prefetching;

4. Query;
5. Result;
6. Search; and
7. Engine

3.1.2.3 Dynamic user profile (DUP) module

The dynamic user profile module creates and maintains the users' dynamic profile. The word extractor maintains the accessed keyword lists extracted from the title tag of a web page, after a click action by a user to reach a new page. This module runs after every click action and starts to extract the words to builds up a keyword list which reflects the user's preferences or interests on the web. The keywords are added to increment the existing counter in the keyword lists. If the keyword is a new word, it is added to the database. If the keyword already exists, it increases the weight of the keyword. The keyword list contains the keywords, the respective number of hits and the last accessed time. This keyword list is used as an input for the predictive module.

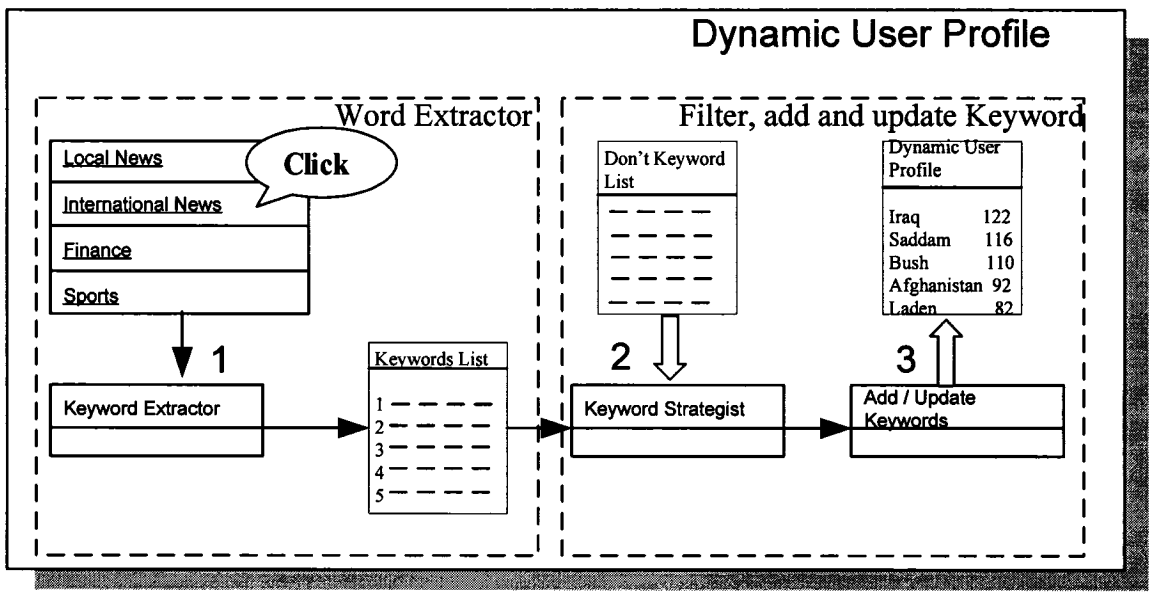

Figure 3.10: Dynamic user profile (DUP)

1. The "keyword extractor" extracts the words from the title tags of web pages when a user browses them;
2. The unwanted words are filtered from extracted the words by using the keywords strategy and "don't keyword" list;
3. The filtered keywords are added to the Dynamic User Profile database if the keywords are new, or update and increase the weight if they are is already in the database.

3.1.2.4 Static user profile (SUP) module

The user static profile is a simple database that contains the user's words or interests and their corresponding ranking. Users can create this profile by entering their words of interest and each word's corresponding rank into the static profile or it can be created and updated by Dynamic User Profile (DUP). The profile manager can add an existing keyword from the dynamic profile to the static profile when a dynamic profile's keyword reaches its highest threshold value. The keyword rankings are positive values, and range between lower (+0.1) and higher (+1.0). A negative value indicates that the words are no longer of interest and the user doesn't wish to prefetch links associated with this word. If the user doesn't want to prefetch any words then he/she can put the word into the profile with a higher negative value.

The profile manager maintains the user's dynamic and static profile database (user's words of interest database) by adding, removing and re-ranking the words of interest (keywords). This static profile database is built in two different ways: first, by manually adding the user's words of interest into the database; and second, by dynamically adding or updating the keywords from the dynamic profile, which are increased to a higher threshold value. If a dynamic profile keyword's rank is greater than the threshold value, then it is added to the user's static profile database

Again, the static profile word's rank will be decreased if the word is not used for a certain number of days. Once the word's rank decreases below the lower threshold value it is removed from the User Static Profile (USP) database.

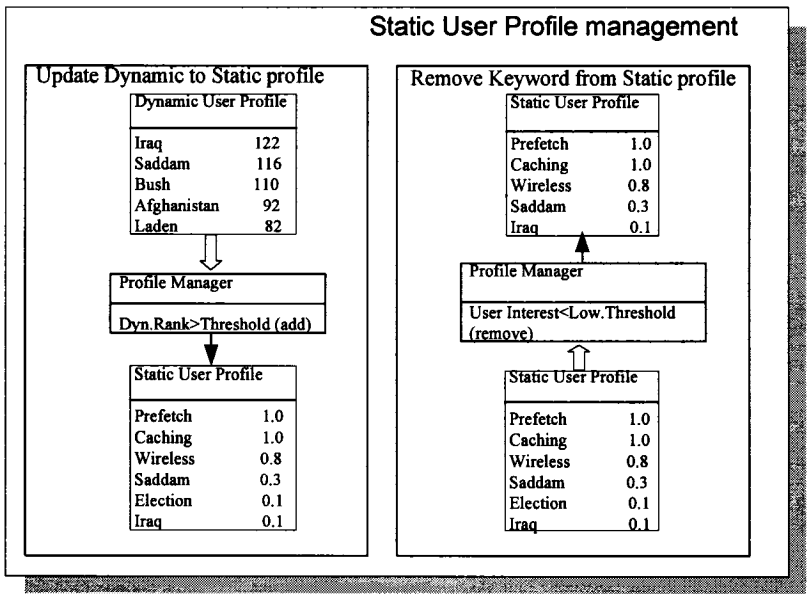

Figure 3.11: Static user profile (SUP) module

3.1.2.5 Neural network training and learning module

The performance of the neural network depends on the network's learning ability. The learning process adjusts the connection strength between neurons using the "supervised" and "unsupervised" learning processes. In our neural network design, we used the "supervised" learning process, where the network adjusted the weights until an expected output was achieved. Back-propagation is a popular learning technique used for the "supervised" learning algorithm. With the back propagation technique the error[1] of the expected result is propagated backwards through the network, and adjusts the network's weights. This process continues until the network error is minimized. In this research we implemented a fixed number of iterations (fifteen hundreds) for this error minimization process. Figure 3.12 shows the basic design diagram of a back-propagation neural network learning module.

Another important challenge of the neural network is the training input data set, which impacts on the network output. The training input data set is presented to the network

with the desired output. The process runs and adjusts the weight gradually until the desired output is achieved.

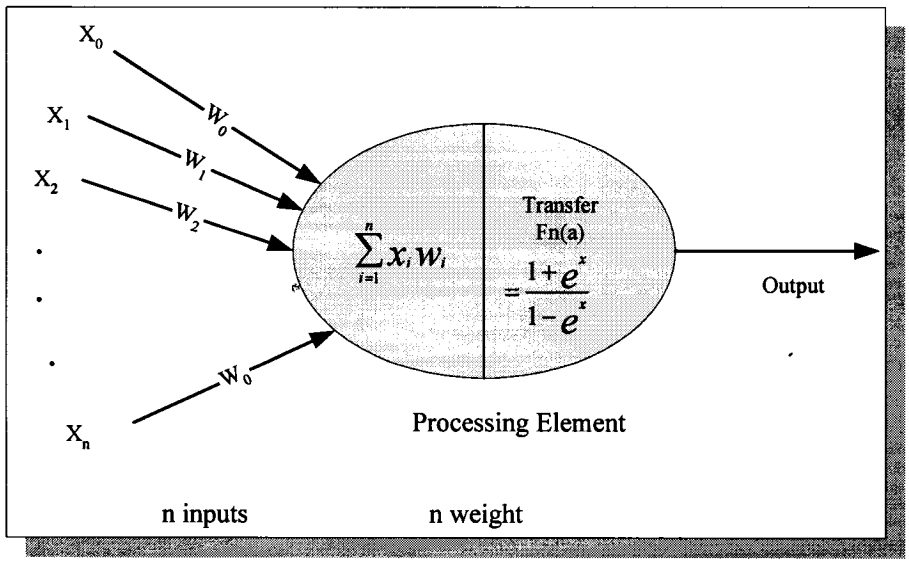

Figure 3.12: Design diagram of neural network learning process

During the training process, the neural network adjusts the weight by using the following formula:

$W_{i,j}(t+1) <- W_{i,j}(t) + \eta.\delta_i.X_i$

i.e. new w_{ij} = old $w_{ij} + \eta.\delta_j.x_i$ (3.1)

where w_{ij} is the weight from the hidden node i to the output node j, δ_j is the error term that we calculated for the output node j, x_i is the output of the hidden node i, and η is a constant value called the learning rate which is a small decimal value (typically 0.07 or 0.025).

3.1.2.5 Prediction module

The prediction module is an important module used for determining if the URL is prefetchable or not. The prediction module is based on the artificial intelligence neural network. The inputs of this module include the rank of the word in the static and the dynamic user profiles, the size, the type and the age of the links, as well as trained net weights.

The neural network calculates the priority (output value) for the given keyword (corresponding URL) by using the different source of inputs as was described above. If the calculated value is greater than the threshold value the neural network sends it to the prefetch manager in order to fetch fetching the link.

$$Input_{BaseVector} = \{URL1, URL2, URL3, \ldots, URLn\} \quad (3.2)$$

$$Word/URL_{InputVector} = \{URL, <X1, X2, X3, X4, X5, \ldots\ldots, Xn>\} \quad (3.3)$$

where:

X1: Word Rank in Dynamic profile (range 0.1 to +1.0)

Ranking = Dynamic keyword's weight (DKW)/m;

where "m" is a certain number of dynamic keyword counters

Assumption: Dynamic keyword weight =m if DKW >=m

The DKW is calculated from the dynamic keyword maker module.

X2: User Static Word Ranking (Input range +0.1 to +1.0)

The word of interest rank is provided from the User Static Profile module.

X3: Object Size (Input range +0.1 to +1.0)

Assumption: Lowest value higher ranking

This value is determined from the keyword parser Module.

X4: Object Type - image, text, application etc (Input range +0.1 to +1.0)

This value is determined from the keyword parser module.

X5: Object Age (Input range +0.1 to +1.0)

Object type input comes from the keyword parser module.
If any of the inputs are not available then set a default value +0.1.

3.1.2.7 Prefetch module

The prefetch module preloads the predicted URLs according to their prediction priority. A higher threshold value URL is selected first from the queue lists and then starts the downloading process. It will stop all on-line prefetching activities when the user clicks on a link or requests a page. If the preloading operation is in ongoing mode then the partially downloaded page will be stored in the cache by sending an "Accept-Ranges: bytes" response header. The next time a user requests that prefetchable page, it will download the rest of the page.

3.1.2.8 Control of the prefetch cache and keyword list of dynamic profile

As we know from section 3.1.2.3 a number of words are parsed from a single URL and placed into the user dynamic profile database. It is not possible to obtain 100% perfection of the keyword list even after filtering the unwanted words. There could be a numbers of words present in to the dynamic user profile that should not be considered as keywords. These unwanted words have a negative impact on the prefetching system. This could result in undesired links preloaded into the cache. Therefore, it is important to build a dynamic user profile of keywords that is totally free of trivial words. Figure 3.13 shows a basic technique that keeps the keywords list up-to-date. If a keyword is not used for a certain number of days (for example, five days), then the rank of the keyword decreases and it continues to decrease in rank if it is not used further. If the word reaches the threshold value T_{Lower} then the word will be purged from the list, as would the associated objects from the cache. The rank of the keyword is calculated with the following formula.

$$WordRank_{new}=(WordRank_{current} - WordRank_{current}*0.1) \quad (3.4)$$

if the word has not been used in the last five days.

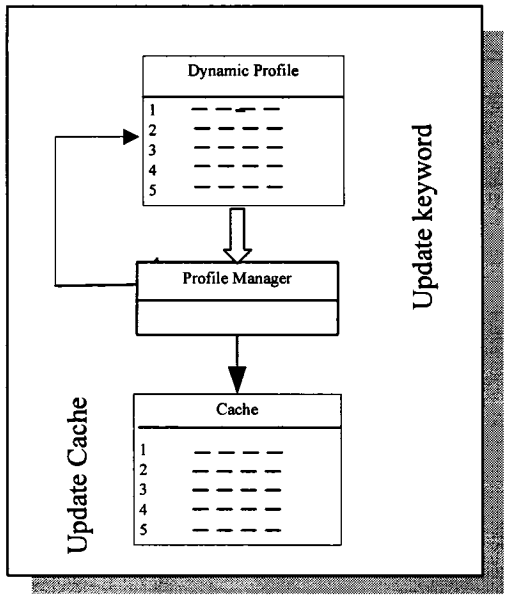

Figure 3.13: Dynamic user profile and cache control

1. Profile Manager calculates the keyword ranking by using formula 3.1.2.8 during the time Proteus started, or any time it is requested by the user.
2. The keyword's rank is compared with the lower threshold value T_{Lower}
3. If the word rank is less than the lower threshold value remove the word from the dynamic profile and delete the cache that is associated with this word.

This technique improves the predominance hit ratio and also removes unnecessary cache from the cache storage, making more space available.

3.2 PROTEUS Software Design

3.2.1 Personalized Artificial Intelligence Proteus Classes

In this section, we will present the software design of Proteus. At first we will describe the system analysis and design of the most important classes especially graphical user interface, caching, prefetching including the neural network based prediction algorithm, and the interdependence between individual classes. Proteus is designed and implemented based on the object-oriented JAVA programming language.

3.2.1.1 The Proteus main class

The class Proteus contains the start point of the main system through the main () method, like any Java program. The main () method instantiates the MainWindow class which is an extension of the java.awt.Frame class. When the MainWindow class is instantiated, it receives a configuration file as a parameter that contains all the information necessary for starting the system: Proxy port, connection timeout, configurations file location, cache and databases information, current caching and prefetching strategies, and cache space size.

```
public class Proteus extends Object {
// Public instance methods
public static void main(java.lang.String[])
```

Class definition 3.2.1.1: Proteus.class

3.2.1.2 The graphic user interface class

The Graphical User Interface (GUI) is the interface between the user and the Proteus system. The GUI was made using AWT, and it provides all administrative activities such as Proxy start, stop, strategy selection and customization of Proteus are made through this interface.

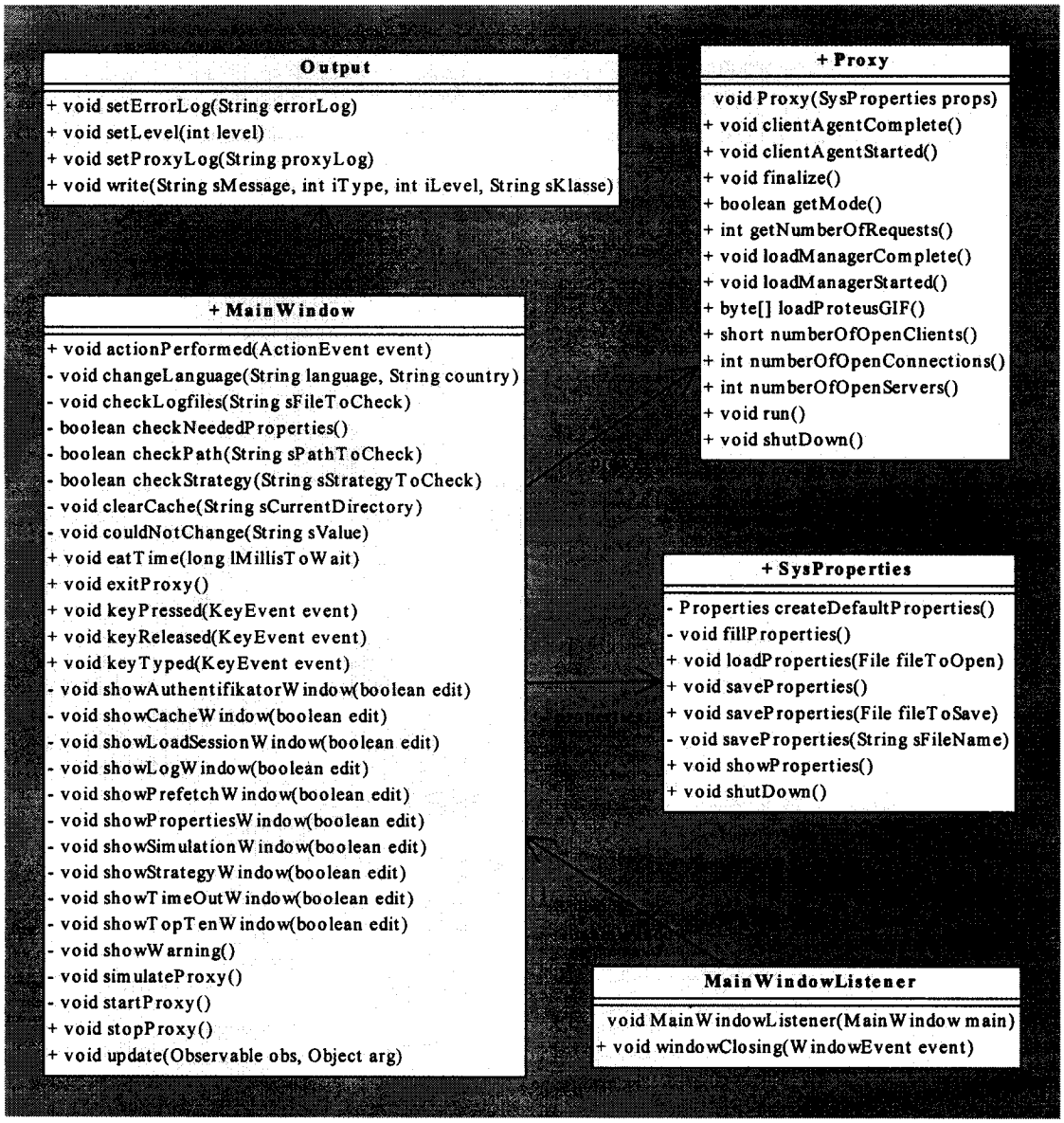

Figure 3.14: UML diagram for graphic user interface

This interface is also used for administrating and managing the Proxy configuration functionality, including language selection, usage and performance statistics, and database management. There are three major classes that are involved in this module: MainWindow, SysProperties and Output classes. Figure 3.14 shows other classes in this module are used for user inputs.

MainWindow class:

The MainWindow class implements the actual user interface. The user can monitor the current status of the system including total cache size and available cache space, number

of objects cached, number of cached hit, and the mode of the system including running, ready or stop mode as shown in Figure 4.1 (a). The user can also control the system including system configuration, database configuration, output configuration, and statistical status reports using this main window. This window also has an execution button to simulate the system. The MainWindow class maintains other menu, submenus, and buttons as displayed in Figure 4.1 (a). This class also acts as a key and mouse event listener for all these menus and buttons.

The buttons "Run", "Simulate", "Stop" and "Exit" Proteus call the corresponding methods startProxy(), simulateProxy(), stopProxy() as well as exitProxy() for manipulation of the Proxy threads. The Check () method is used for validation of resources. The CheckPath() method examines whether the indicated path exists and makes sure the read and write operations are going successfully. The CheckStrategy () method checks whether the indicated strategy class exists and is functioning properly. The method clearCache () method deletes all the cache contents and the directory structures from the current cache location. Language buttons such as "En" for English, "Fr" for French, "De" for German and "Ar" for are Arabic are also implemented through this class.

SysProperties class:

The SysProperties class contains all system settings including default configuration definitions such as path definition, configuration and log files definition, and caching and prefetching default configuration definitions for the Proxy server. During the initialization process, objects of this class are loaded with their default values. The system properties load the attributes from the configuration file and save them to the same file when required. The loadProperties() method is used for the loading of configuration files and other system properties and the saveProperties() method is used for the saving operation to output files or streams. The method createDefaultProperies() creates the default properties values if there are no configurations available.

Output.class:

The Output.class performs and controls the write operation for the output log files and for the console displays that are generated from the individual classes. Write() method is the

main method of this class which operates the write operation on the log file and standard output console. This class behaves like a logger, having different log levels. If a request to log has a lower level than the preset log value it will be discharged. This allows for easy debugging while at the same time not slowing the system with many write operations in a deployed system.

Proxy.class:

The Proxy.class is the main program for a Proxy system. The constructor creates a Proxy and sets the properties received on construction as the properties of this Proxy. This class extends the thread class and the run method triggers the necessary modules and methods to run this program. It creates the authenticator, the cache, the prefetcher and the request queue and then it wait for clients' connections on the port it runs on. The methods showStatus () is used to query the status of the Proxy, including whether the proxy is in operation or down; the total number of requests processed, total number of cache hits, number of objects in the cache, etc. The simulateProxy() method is used to create a simulator for the Proxy with a predefined number of requests.

StrategySelectionWindows.Class:

The StrategySelectionWindows Class is the main class for strategy selection, used for both caching and prefetching algorithms, as well as queuing management. StrategySelectionWindows extends java.awt.Frame and implements the interfaces java.awt.event.ActionListener and java.awt.event.ItemListener in order to intercept the input events.

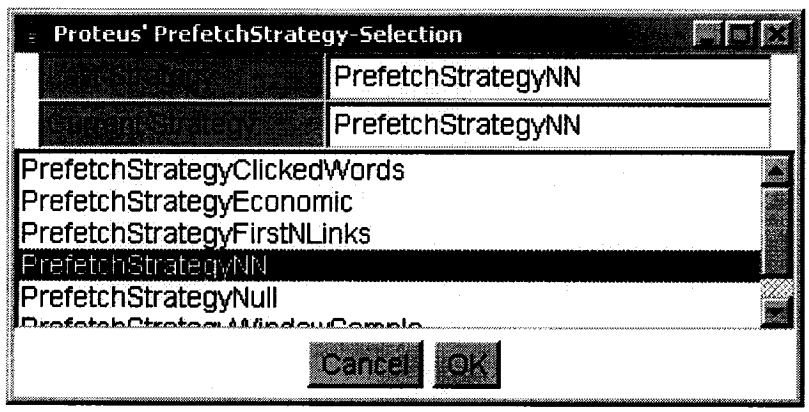

Figure 3.15: Prefetch strategy selection window

This class communicates with the MainWindow through the Observer – Observable pattern. The StrategySelectionWindows class then uses a StrategySelectObserver class that implements an Observable. The only goal of this Observable is to notify the MainWindow of changes in the strategy selection.

PropertiesWindow.class:

The PropertiesWindow.class creates a window that allows for the configuration of the system properties such as the path of configuration files, Proxy port, authentication requirements, etc. A numbers of different Proxy related configurations such as main menu configuration can be done from the "Configuration" menu.

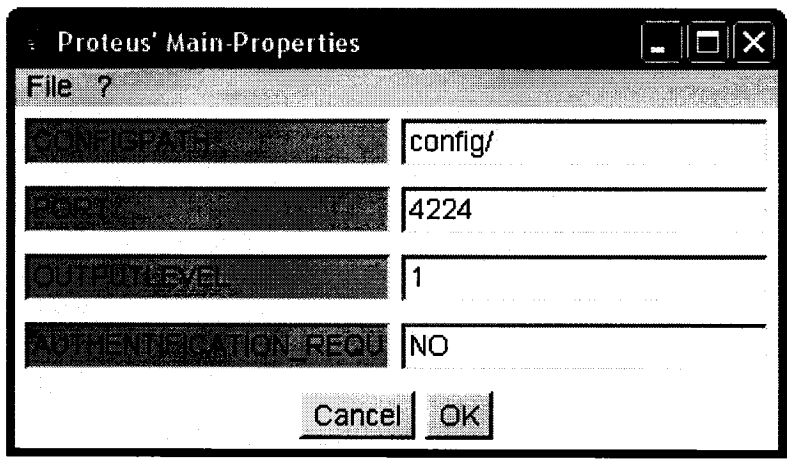

Figure 3.16: Main properties window

Like the StrategySelectionWindows class this class implements the ActionListener and ItemListener interfaces in order to receive the events from the buttons it contains. It also uses an Observable – PropertiesChangedObserver - to notify the MainWindow of changes in the configuration.

Each of the Windows classes also has a Listener class. This Listener class is responsible for any clean-up action required when the window is closed. This is done by extending the WindowAdapter interface and implementing the windowClosing method in the interface.

3.2.1.3 Cache module classes

The cache module consists of objects of the classes that are illustrated in Figure 3.2.1.3. The cache class is the main class for web caching, which controls all the cache operation through other classes. Some important classes that are related to the cache class are CacheProperties, which is responsible for cache configuration; CacheTable, which is responsible for storing HTTP objects' properties for the cached objects; CacheFile, which is used for cache management such as cache addition, deletion and cache validation; and CacheCleaner, which is responsible for cache evictions. In the following section we will describe all other classes in details.

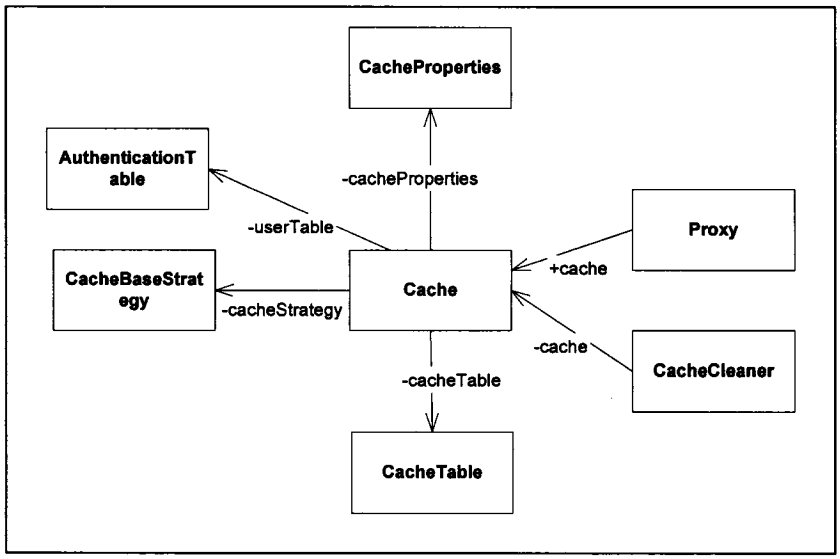

Figure 3.17: UML diagram of cache module

Cache.class

The cache class is the main component of the cache module, and it controls all other classes that are related to the cache module. It is instantiated by the Proxy object and stopped when the Proxy is stopped. During the Proxy initializing, the cache database is also initialized, and it is examined to see whether all indicated objects are actually present in the storage medium. Beside the cache database, the DontCacheVector, which contains the list of object types that are not to be cached, is also initialized during the Proxy start

up. The CacheFile() method takes care of all basic cache operations, including caching a new object, updating cache and validating cache by calling the method cacheNewFile(), UpdatedFile() and UpdateElement() method respectively. CacheCleaner is called if the cache needs to delete from the cache storage.

CacheProperties.class

The CacheProperties class is responsible for the configuration of cache relevant attributes, which are related to the cache operation and management. The following configurations can be made by this class when they are required:

- The name of the configuration path, in which the configuration files are stored
- The name of the cache database
- The name of the DontCache database
- the name of the log file in which all cache events are logged
- The name of the caching strategy
- The maximum cache size in bytes.

CacheTable.class

The CacheTable class implements a hashtable, in which the HTTP objects' properties are stored for the cached objects. The method addElement() takes care of adding new objects into the table, removeElement() deletes objects from the table and getElement() returns the HttpFileProperties object for the relevant operations.

CacheFile.class

The CacheFile class takes care of physical file and folder operations such as addition and eviction of cached objects from the cache. The method createFileName() determines under which name the object is to be stored. The method delete () deletes the current file and calls deleteDirectory() afterwards to delete the directory, if it is empty. The method load () loads the web object and save () stores the files in the local path.

CacheStrategy.class

The cache strategy class specifies which objects are to be stored in the cache and which objects are to be cached or removed from the cache. This class sets the priority for all

cached objects and assigned a TimeToLive value, which indicates how long the file is to be stored. The two main classes the CacheStrategy class and the CacheStratKnownValues class are responsible for cache strategy management.

CacheCleaner.class

The CacheCleaner.class is responsible for cache evictions. In Proteus, ten different cache evection algorithms are implemented, including artificial intelligence neural network. Those algorithms are used for deleting the files from the cache. The method startCleanAlgo() uses a SWITCH instruction to start the desired erasing process.

3.2.1.4 Prefetch Class:

The main function of this class is to predict and download web objects from the Internet before they are actually requested by the user, by considering the user's web interests and the object's important cacheable properties such as type, size, age, etc. There are some other supporting classes available in this class - mainly the PrefetchManager, PrefetchBaseStrategy, ClientAgentProperties and PrefetchProperties. The supporting classes are shown in the UML diagram shown in Figure 3.2.1.4. PrefetchManager manages the whole prefetch system, including administration and coordination of the prefetch and prediction strategy. In the following section we will describe the related classes in detail.

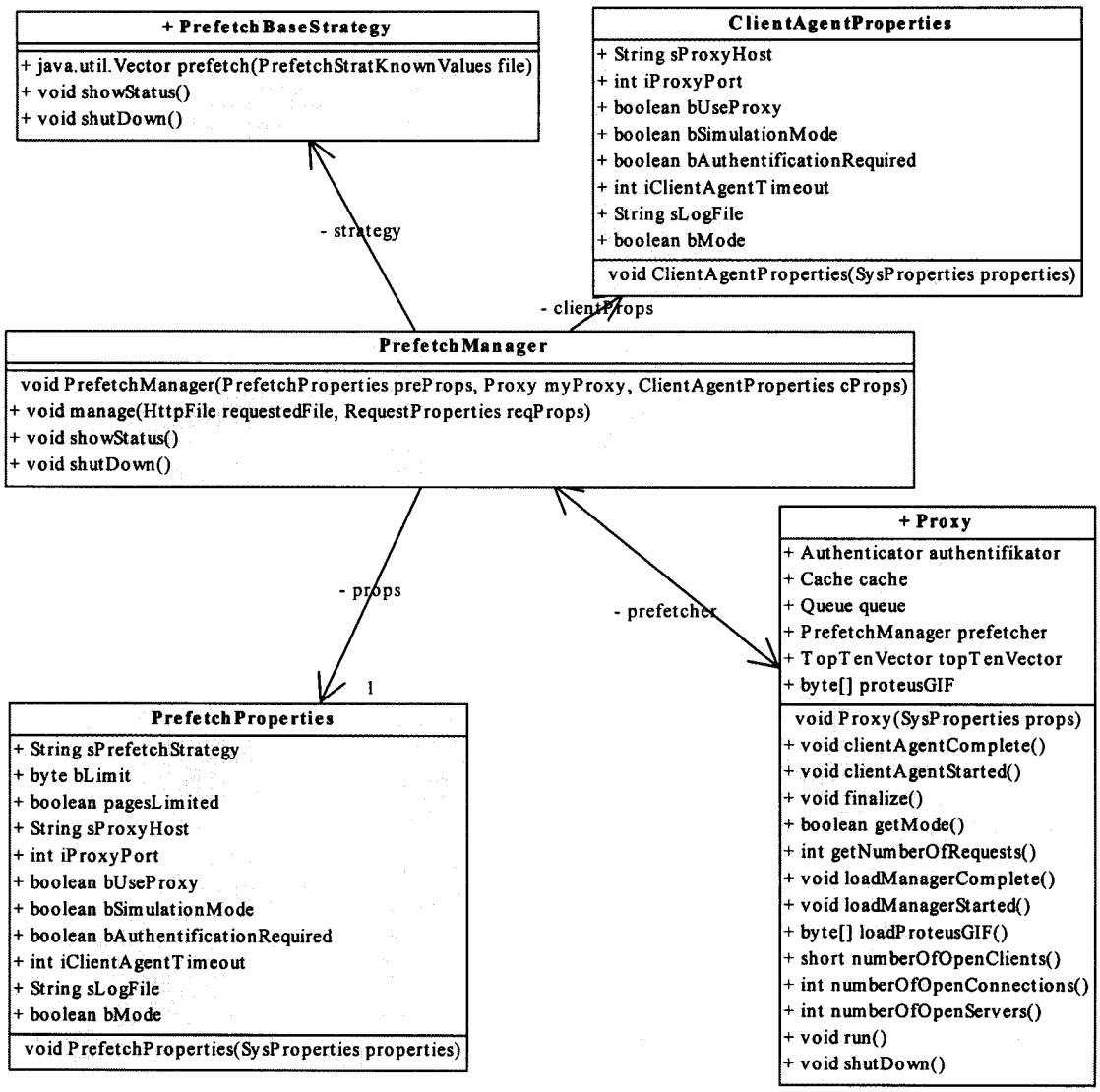

Figure 3.18: UML diagram for prefect module

PrefetchManager.class

The PrefetchManager class is the main class for managing and controlling the prefetch strategy. It is instantiated directly by the Proxy system during the initialization of the Proxy. During the instantiation, the PrefetchManager selects a strategy class and coordinates the strategy. The method manage () method controls the preloading of links and communicates with the MainWindow class.

The method showStatus() displays the current status of the prefetch strategy. The method showStatus() can be called from the menu "Tools" at any time during Proteus operation or non-operation mode.

The shutDown() method is called when the program is terminated or upon the completion of the program. This class ensures that all the strategy data is sent and received before closing the program.

PrefetchProperties.class

The PrefetchProperties class sets the values of prefetch parameters and the classes required by the PrefetchManager. These values are:

- PagesLimited – an indication of whether there should be a maximum number of links which can be loaded;
- PAGE limit - the limit of pages to be loaded, if PagesLimited is true; and
- PrefetchStrategy - the strategy to be used.

PrefetchStratKnownValues.class

The PrefetchStratKnownValues class stores information about an HTTP file, the request properties of that file and the cache of the Proxy. This is an interface between the strategy and the classes of the Proxy framework.

PrefetchBaseStrategy.class

The PrefetchBaseStrategy is the base strategy for developing other prefetch strategies. All additional strategies must extend from this class.

HTMLLinks.class

The class is used for collection and determination of link properties. Initially, an HTML link object consists of three public instance variables:

- *public String sUrl* - stores the name of the URL of the reference.
- *public String sText* - stores the words which represent this URL on the web page.
- *public int iScore* – used for weighing the links and sorting them

HTMLPageParser.class

This class analysis the HTML tags of a web document and extracts the page or link and associated text of the HTML page. Only the public method getLinks() is available which produce a vector of HTMLlink objects.

MultilayerPerceptronLearn.class

This is the only class that is used for the neural network's training and learning process. It takes different data inputs, including NumberOfEpochs which represents the number of iterations, NumberOfInputs which represents the number of network inputs, NumberOfHiddenLayer, which represents the number of hidden layers, NumberOfRecords, which represent the number of training sets, and NetworkLearningRate which determines how much the weight will be changed in every step. The calcNet() method reads the training data set, including a bias input from an input file, and calculates the weights until the required output is reached. WeightChangesHO() and WeightChangesIH() adjust the weight of hidden-output and input-hidden, respectively. The calcOverallError() method calculates the overall network error after each epoch. tanh() is the S-shaped hyperbolic tangent (tanh) function which is used for error minimization. The weightWrite() will write the weight into the result.dat file. This class does not need to be run every time Proteus is started but it is needed when a new training data set is available.

MultilayerPerceptronPrediction.class

This is the main class for prefetching prediction. The readWeights() method reads the training weights from the file result.dat. The inputNeuronValue( ) method reads the inputs from the parser class. The calcOutPuts() method calculates the network output value. If the calculated value is larger than the threshold value, then it sends it to the prefetch class.

HttpRequester.class

This class retrieves the header information from an HTTP file. An HttpRequester is constructed for every file that we need to retrieve the header from. Then, the getInfo() method is invoked, which will send a request for the header of the HTTP file. The response is then parsed and the necessary information is retrieved from it: content type, content size, last modified date and content encoding. These properties can then be retrieved using getter methods.

InvalidKeywords.class

This class is responsible for reading the configuration file that contains words that should not be keywords, and adding them to a hashtable. The only method that it supports is the method isValid(), which will return true if the passed parameter is a valid keyword and false otherwise.

KeywordSelector.class

This class is responsible for scanning the anchors of links that have been retrieved and removing the words that should not be keywords. It receives a hashtable at instantiation that contains links as keys and anchors as values. When the call to the method findKeywords() is done, it goes through the hashtable, tokenizes the anchor strings, and generates an array list of keywords for this link by removing any words that are less then 4 characters and that are not keywords.

LinkInfo.class

This is a simple class that stores information about an HTTP file such as link location, keyword array list, encoding type, content type, size, and the date last modified. It provides getters and setters for all these properties.

LinkOrganizer.class

This class is the main coordinator of the subsystem. It receives a URL and then uses the other classes to parse the page located at that URL, obtain the links in that page, get

information about those links, compute keywords and generate an ArrayList of processed LinkInfo objects. It also provides a getter for the LinkInfo objects.

NNValueCalculator.class

This class is responsible for calculating the NeuralNetwork value of a link. It receives a LinkInfo object on instantiation and computes the NN value for each of the attributes. It then provides getters to obtain the NN value for size, encoding type and date last modified.

WebPageParser.class

This class parses a web page to obtain the links and the relevant anchor information for those links. Once this is done, it provides a getter that retrieves an ArrayList of links and anchor values.

UserProfile.class

This class is responsible for reading the user profile from a file and then generating from the profile a hashtable that stores strings representing keywords as keys and NN values for those keywords as hashtable values.

CacheNN.class

This class is responsible for using the NN module to determine the caching strategy. It is an extension of the CacheBaseStrategy class and implements all of its methods.

Chapter 4: Implementation

There are three important applications integrated with the current version of Proteus as described in chapter 3. These are web caching, web prefetching, and caching and prefetching simulator. The following sections provide the implementation of each of the applications mentioned above with different algorithms.

The two major components, caching and prefetching, are integrated with this current version of Proteus. The user interface of Proteus is implemented in various languages so that it can be configured and monitored through any language, as shown in Figure 4.2. This version of Proteus is also very user friendly due to its easy implementation and configuration. Proteus is platform- and client browser-independent so that it can work with any operating system and any popular web browser. In the next sections, we will describe the Proteus implementation in detail.

4.1 Basic Proteus

Proteus is implemented using the Eclipse SDK project's object-oriented programming language with JAVA technology. The user interfaces and all Proteus functionalities, such as web caching, prefetching, cache management, strategy management, database management, etc., are implemented with JAVA technology. The different databases and configuration files are implemented using an Operating System (OS) based simple flat file system. Figure 4.1 shows the simple graphical user interface of Proteus. This interface contains different buttons used to start and stop the system, the "Languages" button for changing the language, and the "Simulate" button for simulating the system.

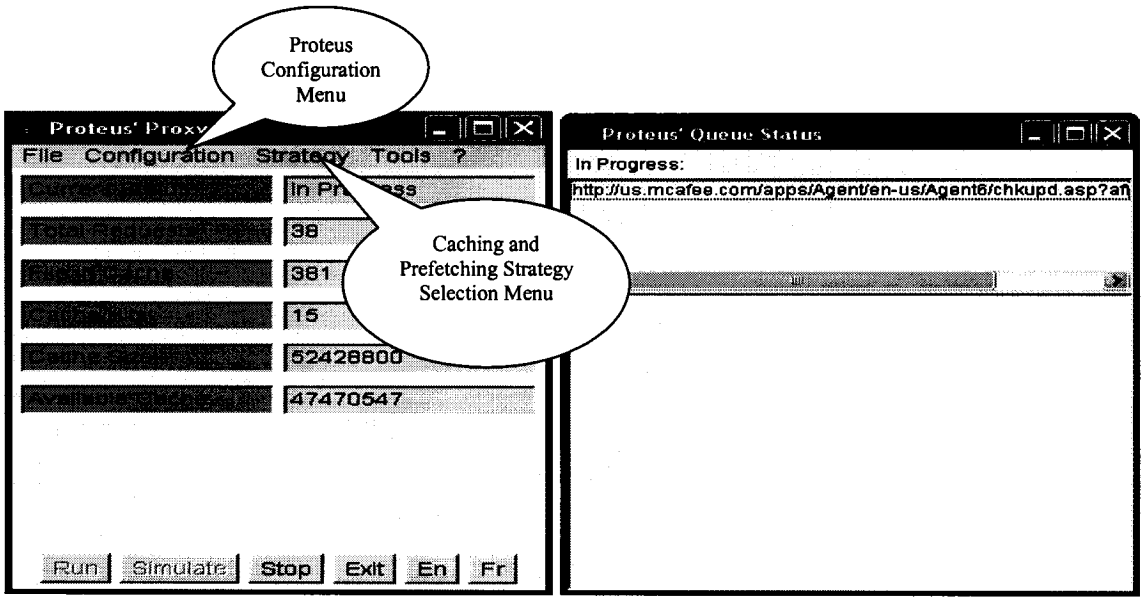

Figure 4.1: Graphical user interface of Proteus.

Proteus' standard menu items, including File, Configuration, Strategy, and Tools are used for file management, configuration of the Proteus system, strategy selection, and monitoring the status of Proteus and the statistics of caching and prefetching. In Figure 4.1, Proteus' "Queue Status" shows an important utility used for monitoring the queue status of Proteus.

4.2 Internationalization Proteus

In software engineering, internationalization is an important feature that adequately adapts software for use in various languages and locales. Software internationalization is defined as a way of designing and producing software that can be easily adapted to any person in any region. The internationalized software has the required flexibility to implement an application with minimal cost and effort and without changing source codes. In this project we implemented a multilingual version of Proteus, which works with English, French, German and Arabic, but can also extend to any other language.

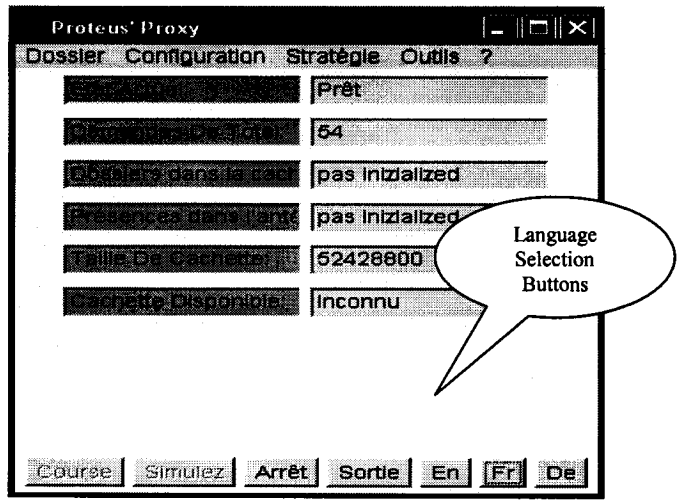

Figure 4.2: Proteus with French menu

The language is selected using the language buttons located at the bottom of the main window of Proteus. Figure 4.2 shows the main menu of Proteus using the French language.

4.3 Prefetch Strategy Implementation

There are a number of different strategies implemented in Proteus for caching and prefetching purposes including CacheStrategyEconomic, CacheStrategyEconomic, CacheStrategyEstimateFileSize, and CacheStrategyEstimateTimeSinceLastRequest for caching, and PrefetchStrategyEconomic, PrefetchStrategyClickedWords, PrefetchStrategyFirstNLinks for prefetching. In this new version of Proteus, we implemented our latest artificial intelligence neural network based on CacheStrategyNN for caching and PrefetchStrategyNN, PrefetchStrategyNN1Input, PrefetchStrategyNN2Input, PrefetchStrategyNN3Input for prefetching strategies. It is necessary to change the different strategies in order to compare the performance of individual strategies. Proteus is designed and developed in such a way that there is no need to search the source code in order to rewrite the software for a new strategy. A new caching or prefetching strategy can apply to Proteus through the prefetching strategy selection windows as shown in Figure 4.3.

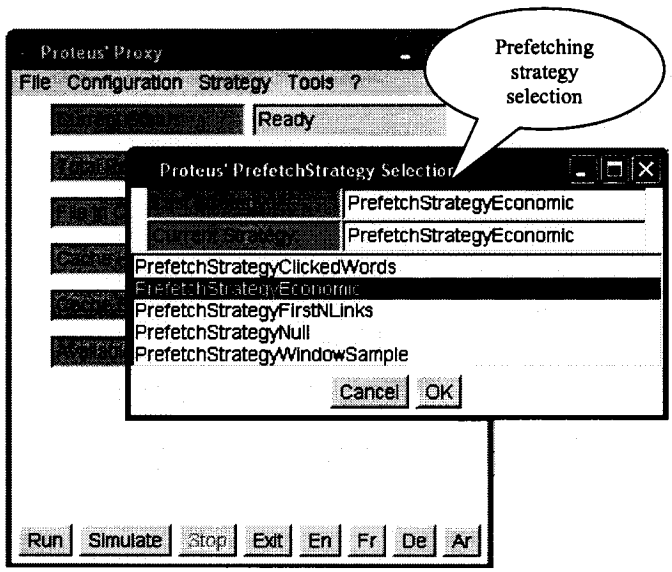

Figure 4.3: Prefetch strategy selection window

A new strategy can also be applied or selected by editing the /config/proteus.cfg configuration file. The new strategy only activates after restarting the Proteus system.

4.4 Artificial Intelligence Prefetching

The main purpose of the prefetch module is to determine if web links are prefetchable or not. If the link is prefetchable, then the AIMNN (Artificial Intelligence Multilayer Neural Network) sends it to the prefetch manager for downloading; otherwise, it tests the next link. The prefetching prediction has been implemented through three sub modules: the "Multilayer Perceptron (MLP) training module" used for network training, the "prediction module" used for prefetching prediction, and the "prefetch manager" used for preloading the predicted links.

The training module trains the network using a training set /config/InputX.nn. This training module is very important for the artificial neural network because the final network output result depends on this trained network's output. The training module writes the trained network to an output /config/network.nn file. The sub-module "neural network" of the prefetch module uses this trained network (recall) for computing the output for a given input. The prediction module then compares this network output with

given prediction threshold values and then forwards it to the prefetch manager if it satisfied by the prefetch threshold value for downloading the predicted objects for future use. To get a better prediction result from the neural network, it is necessary to select a good training set, number of network layers, network learning rates, optimized network input values and prediction threshold value. The following sections describe the implementation of each of the terms mentioned above.

Neural network training set:

In the implementation of our MLP training algorithm, we have used five different training sets corresponding to five different inputs. A sample training set is shown in the Table 4.1 with two inputs (first and second columns) and an expected output in column three. We have trained the network with five different training sets and saved the trained output for further use in the neural network in the prediction module. In our training set (/config/inputX.nn) we used different combinations of network inputs (word ranking in static profile, word ranking in dynamic profile, size, type, age, and expected downloading time).

Input X: Word Ranking in Dynamic Profile	Input Y: Word Ranking in Static Profile	Expected Output: = (120X+90Y/210)
0.1	0.1	0.100000
0.1	0.5	0.271429
0.1	0.8	0.400000
0.1	1.0	0.485714
0.5	0.1	0.328571
0.5	0.5	0.500000
0.5	0.8	0.628571
0.5	1.0	0.714286
0.8	0.1	0.500000
0.8	0.5	0.671429
0.8	0.8	0.800000
0.8	1.0	0.885714
1.0	0.1	0.614286
1.0	0.5	0.785714
1.0	0.8	0.914286
1.0	1.0	1.000000

Table 4.1: Sample NN training set with two inputs.

Keyword ranking in dynamic profile:

The dynamic profile keyword rank is built and managed through the parser module, which was discussed in section 3.1.2.2. A sample of the dynamic keyword ranking and its associated neural network values are shown in Table 4.2. Table 4.2 is a Grade 5 student's web interest ranking as of January 3rd 2006. We implemented dynamic keyword ranking with the following formula:

NN value = current word rank/20. (Where max is +1.0, min is +0.1, and negative values are for removal from the database)

Keywords	Last Access Time	Word Rank (#of hits)	Word Rank (current)	NN value
Canadian	Sat Dec 17 20:12:57 EST 2005	27	21.6	1.00
Government	Sat Dec 17 20:12:57 EST 2005	37	31.6	1.00
Paul	Sat Dec 17 20:34:03 EST 2005	12	6.6	0.33
Martin	Sat Dec 17 20:34:57 EST 2005	12	6.6	0.33
Yu-Gi-Oh	Sun Jan 01 10:24:56 EST 2006	123	123	1.00
Classroom	Sun Jan 01 10:54:16 EST 2006	18	18	0.90
Jokes	Sun Jan 01 00:54:16 EST 2006	241	241	1.00
Armour	Fri Oct 22 18:43:15 EST 2005	32	-5.8	0.10
Medieval	Fri Oct 22 18:43:15 EST 2005	42	4.2	0.21
Pokemon	Fri Oct 22 18:43:46 EST 2005	189	151.2	1.00
Soccer	Fri Oct 22 19:04:47 EST 2005	39	1.2	0.06
Governor	Fri Oct 22 18:43:21 EST 2005	2	-3.4	- 0.10
General	Sat Dec 17 20:12:57 EST 2005	2	-3.4	- 0.10

Table 4.2: Keyword ranking in dynamic profile

Keyword ranking in static profile:

The static user profile contains the user's interest keywords associated rank and associated neural network value as shown in Table 4.3. We implemented a static user profile in a flat file /config/staticProfile.dat . Table 4.3 shows a sample profile for the same grade 5 student.

Web Interest	Keywords	Rank	NN Value =Rank/Highest rank
Canadian Government System	Government	5	0.5
Weather and Climate	Weather	5	0.5
Classroom Jokes	Jokes	10	1.0
Silly Jokes	Jokes	10	1.0
Fiction Story	Fiction	8	0.8
Unfortunate event	Unfortunate	8	0.8
Harry Potter	Harry	10	1.0
Goose bumps	Goose	5	0.5
Medieval times	Medieval	10	1.0
Pornography	Pornography	-10	-1.0

Table 4.3: Keyword ranking in static profile

Object type:

The parser module determines the object type by sending the header request to the host server. The prefetch module compares the current keyword's object type with the object type table and sets the input value for NN input for each object type parameter. Table 4.4 shows the object type and its corresponding NN value.

Object Type	NN values
Image	1.0
Text	0.8
Application	0.5
Audio	0.2
Video	0.2
Others	0.1
Unknown	-1.0

Table 4.4: Object type and associated NN values

Object size:

The parser module also determines the object size for the current keyword by sending its header request. The ranking module compares the current keyword's object size with this table and sets the input value for NN input. Table 4.5 shows the object size and the corresponding NN values.

Object Size in KB	NN Values
>0 KB < = 10KB	1.0
>10 KB < = 20 KB	0.9
>20 KB < = 40 KB	0.8
>40 KB < = 100 KB	0.6
>100 KB < = 200 KB	0.5
>200 KB < = 400 KB	0.3
>400 KB < = 800 KB	0.2
>800 KB < = 1000 KB	0.1
>1000 KB	-1.0

Table 4.5: Object size and associated NN values

Object age:

Web object modification rates or age are important parameters for both web caching and prefetching. The parser module determines the object's age for the current word by sending a header request. The ranking module compares the current keyword's object age with this table and sets the input value for this NN input. Table 4.6 shows the object size and the corresponding NN value.

Object Size in KB	NN Value
>30 days	1.0
>20 days < = 30 days	0.9
>10 days < =20 days	0.8
>5 days < =10 days	0.5
>5 days < = 10 days	0.3
Otherwise	0.1

Table 4.6: Object lifetime (age) and associated NN values

Neural network training:

In our neural network implementation, we used a Multilayer Perceptron (MLP) algorithm, a very powerful technique discussed in section 2.2.3.2. The MLP based network has been trained with different numbers of layers and different learning rates and

we obtained the results as shown in Figure 4.4 and Figure 4.5 respectively. The results show that the network produced the expected result with four layers and has a learning rate of 0.3. For the remainder of the test and the Proteus implementation, we used a four layer back propagation neural network module with a network learning rate of 0.3.

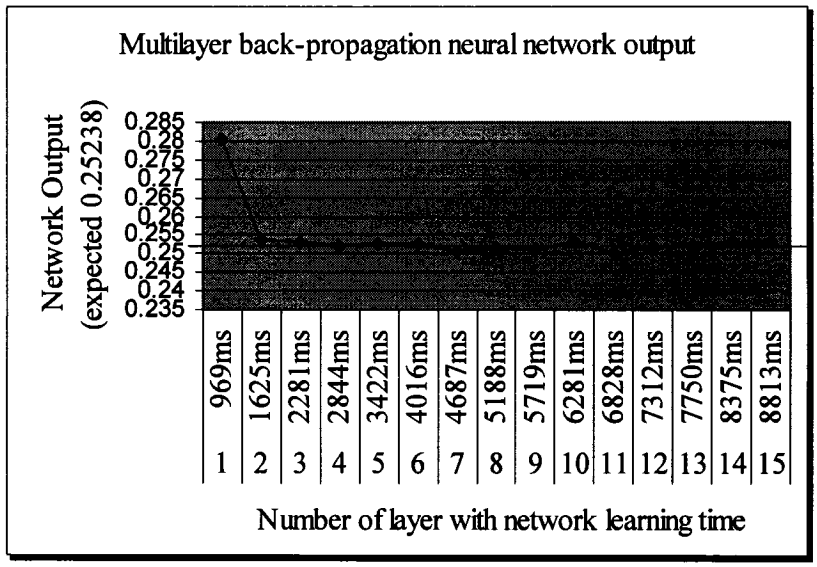

Figure 4.4: Network output with multilayer back-propagation

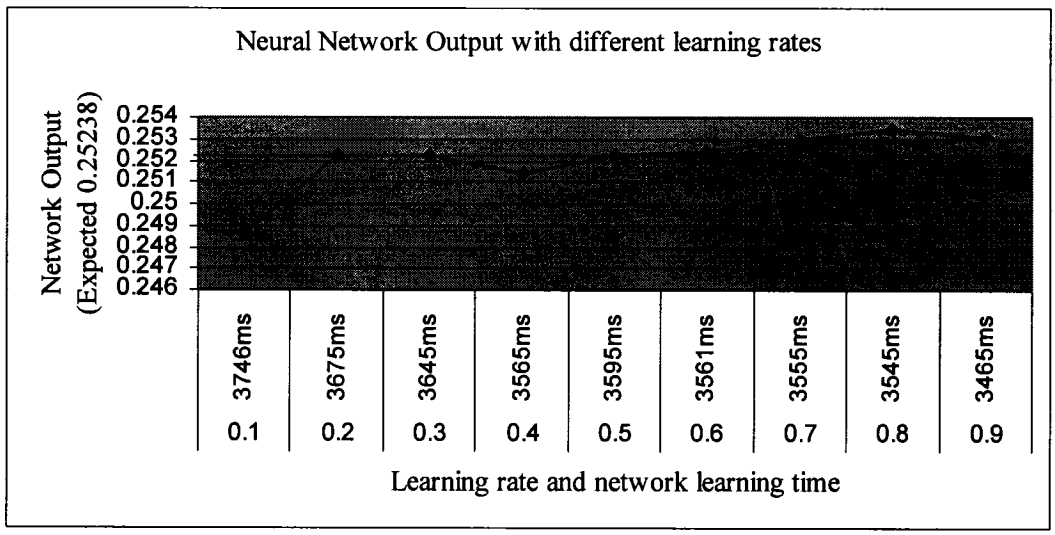

Figure 4.5: Neural network output with different learning rates

Chapter 5: Evaluation and Results

5.1 Testing Environments

To evaluate the performance of the proposed system, we have experimented with Proteus using candidates of various age ranges and professions. These experiments were conducted on a Satellite Pro 6100 model Toshiba Laptop with Intel Pentium 3 running at 1.1 GHz with 256 MB of RAM using Windows XP professional as the OS. In this experiment, several different client browsers such as Microsoft Explorer, Mozilla FireFox, and Netscape have been used to perform this test. For all the test cases, we set the browser cache to zero "0.

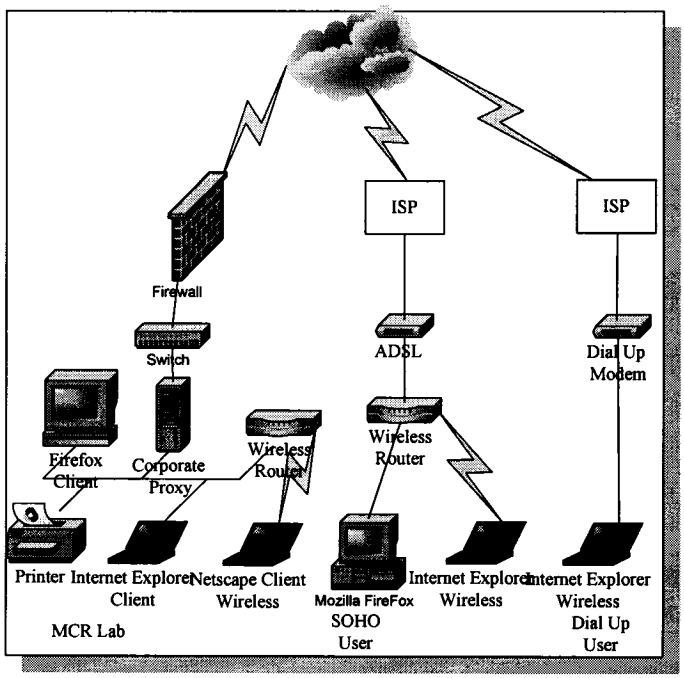

Figure 5.1: Testing environments for Proteus.

We also conducted these experiments in various network infrastructures, including dial up, broadband and a wireless LAN, demonstrating a mobile system. Figure 5.1 shows typical scenarios of the experiment environments, which are described below:

MCR Lab, University of Ottawa, Ottawa:

- Under strict firewall (University firewall),
- University Proxy server
- Microsoft Internet Explorer
- Mozilla Firefox
- Client OS: Windows XP/2000,
- Network: 100 Mbps Ethernet LAN.

SOHO Network, Ottawa:

- Behind wireless router
- Behind a software-based firewall
- LAN and WLAN workstations
- Microsoft Internet Explorer
- Mozilla Firefox
- Using ADSL,
- OS: Windows XP/2000,
- Network: 100 Mbps Ethernet LAN with NAT
- WLAN: 54 Mbps connection
- Two users use Proteus simultaneously in small office and home office environments.

Dial up User, Ottawa:

- Dial up user connected to ISP through modem
- Netscape
- OS: Windows XP.

5.2Test Results

In this section, we analyze the performance of the artificial intelligence Proteus system. Number matrices have been used to test the performance of the caching and prefetching systems. There are two popular metrics: "hit rate" and "byte hit rate" which are frequently used for performance testing. "Hit rate" is defined as the number of requests served from the cache, divided by the total number of requests served. The "byte hit rate" refers to the number of bytes served from the cache, divided by the total number of bytes served. In this study, we emphasized the hit ratio matrices. We compared the "cache hit ratio" with the cache only as well as the combination of caching and prefetching.

To measure the performance of the prefetching system, we have evaluated certain metrics, such as the "prefetch hit ratio" and "waste ratio". The "prefetch hit ratio" refers to the ratio of the total number of prefetched objects that are consequently requested by the user to the total number of user requests as shown in equation 5.2. The second criterion the "waste ratio," is defined as the ratio of the percentage of undesired documents that are not requested by the user to those pre-fetched in the cache. We have conducted the experiments for the performance of Proteus with two different prediction parameters, the "threshold value" and the neural network "learning rates."

Three important terms, referred to as the "number of user requests," "number of predictions" and "number of prefetch documents" are used for determining the prefetch matrices. The definition of these terms is as follows:

- Predictions: number of objects predicted by the prediction module.
- Prefetch: number of objects prefetches by the prefetching module.
- Good Predictions: number of objects predicted that are afterward requested by the user
- Bad Predictions: Predictions that are not afterward requested by the user.
- Objects Not Used: amount of prefetched objects never requested by the user.
- User Requests: total number of objects the user requested.

There is a relationship between the three prefetch terms "user request", "prefetch" and "prediction," as shown in the Figure 5.2 (set).

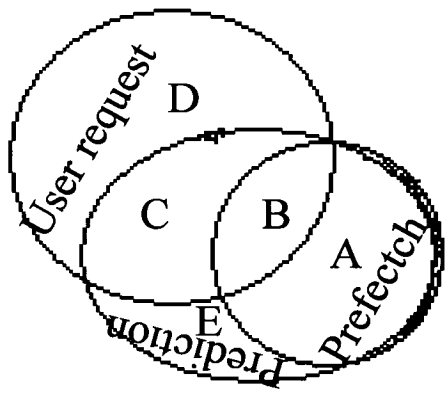

Figure 5.2: Relationship between user requests, predictions and prefetch

In the diagram, "D" represents the requests made by the users that are never predicted or prefetched. "C" represents the prediction that has not been prefetched. "B" represents the good predictions that have been prefetched and requested by the user afterwards. This is called the prefetch hit. "A" and "E" represent the bad prediction of objects which are not requested by the user.

$$prefetchHitRatio = \frac{\text{total number of prefetched objects that are consequently demanded by the user}}{\text{total number of user requests}} \quad (5.1)$$

5.2.1 Experiment Result of Cache Hit Ratio With Web Caching Only

The web cache performance hit ratio is defined as the ratio of successful hits against the total number of requests for any time. We tested the cache-hit ratio for different cache space sizes, such as 1 MB, 5MB 10MB, 15MB and 20MB over approximately a two month period. During the testing period, we used our system in various different environments, such as a wired network, a wireless network, and firewall and non-firewall

environments. During this testing phase, we used several different browsers including Internet Explorer, Netscape and Firefox.

Cache Space Size	Objects in Cache	Session Requests**	Success Cache Hits	Hit Ratio
1 MB	87	213	4	2%
5MB	1095	301	74	24%
10 MB	1749	2428	802	32%
15 MB	2288	1171	398	33%
20 MB	2867	982	282	29%

Table 5.1: Hit ratio for web caching with different cache sizes

The experimental results for the hit ratio with corresponding cache space sizes can be found in Table 5.1 with a graphical representation in Figure 5.3. The first column of Table 5.1 displays the cache size; the second column shows the number of objects cached, and column three displays the session requests made by the user for the specific session. The fourth column represents cache hits. It is clear from the above diagram that the hit ratio increases as the cache size and as the number of objects increase.

We plotted a graph for "cache hit ratio vs. user request," as shown in Figure 5.3, for a 12-day period, starting with a zero cache. Figure 5.3 shows that the hit ratio increases along with requests and after a certain number of requests (at 3000) the hit rate starts to become steady. We obtain an approximate average hit rate of 32% at a cache space size between 10MB and 20MB.

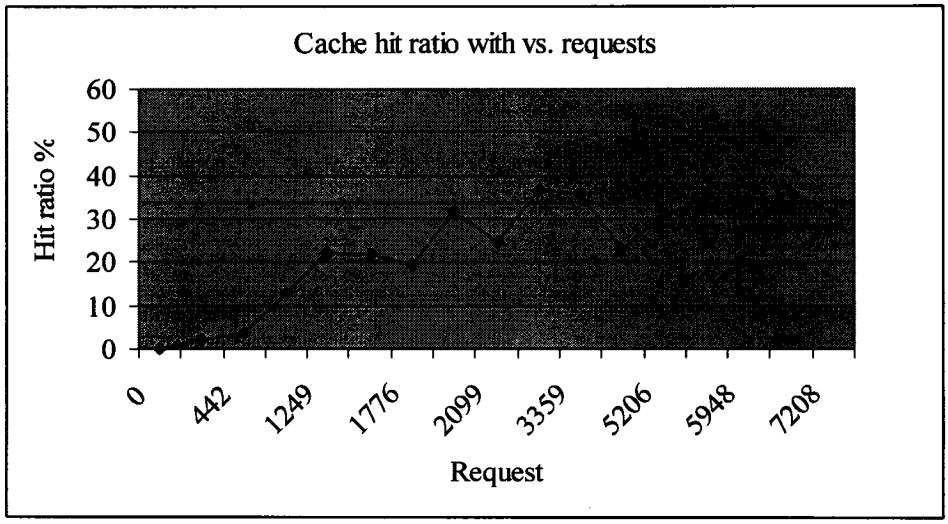

Figure 5.3: Cache hit ratio vs. requests

5.2.2 Cache Hit Ratio Integrated Web Caching And Prefetching

To evaluate the hit ratio for the combination of caching and prefetching, we tested the cache-hit ratio with different cache sizes such as 1 MB, 5MB 10MB, 15MB and 20MB for a 10-day period. During this experimental period, we used our system in different environments including a wired network, a wireless network and firewall and non-firewall environments. We also used several different browser including Internet Explorer, Netscape and Firefox during this testing phase.

Cache Space Size	Objects in cache	Session requests	Success cache hits	Hit Ratio (apprx.)
1 MB	103	176	12	7 %
5MB	1191	723	199	28 %
10 MB	1732	354	151	43 %
15 MB	2167	933	358	38 %
20 MB	2938	367	231	41 %

Table 5.2: Hit ratio with web caching and prefetching as combined vs. cache size

The cache hit ratio for different cache sizes is shown in the first column of Table 5.2. The second column shows the number of cached objects for the cache corresponding cache size. The third and fourth columns represent the session requests made by the user

and the cache hits, respectively. The results of the test show an average hit ratio of approximately 40 % at cache sizes between 10MB and 20MB.

5.2.3 Comparision of Cache Hit Ratio Between With and Without Prefetching

Tables 5.2.1 and 5.2.2 show in both cases that the hit ratio increases with cache space size. The results show that the cache hit ratio is approximately twenty-six percentage higher when the system performs with the integration of caching and prefetching, rather than with cache only. In both cases, we acquired the result using a small cache size and for a small period of time. We believe that the cache hit ratio will improve if the system learns for a considerable amount of time. The performance of Proteus (in both caching and prefetching) depends on the user's static and dynamic user profile, which reflects the user's web interests. Therefore, if the system learned for an extended period of time, the user profiles would be rich in pertinent information, and the system's performance would increase accordingly. We are very confident that the hit ratio can reach around 50% if the system fully learns the user's web behavior.

5.2.4 Prefetching Prediction Time With Different Network Inputs

We evaluated the performance of the prediction system's prediction time with different inputs, as shown in Table 5.3.

Test Case #	Dynamic user Behavio ur	Static User Behavio ur	Object Size	Object Type	Object Age	Download Cost	Network Training[+]	Avg. Prediction Time	Prefetch hit ratio (%) (Approx.)
1	YES	NO	NO	NO	NO	NO	167ms	209ms	48%
2	YES	YES	NO	NO	NO	NO	221ms	287ms	53%
3	YES	YES	YES	NO	NO	NO	861ms	422ms	55%
4	YES	YES	YES	YES	NO	NO	1001ms	1641ms	57%
5	YES	YES	YES	YES	YES	NO	14982ms		
6	YES	YES	YES	YES	YES	YES	63524ms		

Table 5.3: Required time for network training, and prediction

[+] Network training is require only when the training set changes

The first column of the Table 5.4 represents the test cases' numbers; the 2nd to 7th column represent the different network inputs for the prediction unit; column 8 represents the time for network training; column 9 represents the average prediction time including preprocessing and neural network recall time, and column 10 represent the prefetch hit ratio. The results show that the average prediction time increases with the number of inputs along with the prefetch hit ratio.

We are currently also evaluating the prediction timing effect of the networks with 5 inputs (case study 5) and 6 inputs (case study 6) to determine an optimum value for the number of inputs.

5.2.5 Prediction With Different Theshold Value

The system was evaluated, and then we plotted the prefetching prediction with different neuron threshold values of 0.25, 0.50 and 0.75. As seen in Figure 5.4, a higher threshold value produces a lower number of predictions, and there are a higher number of predictions made by the prediction module when using lower threshold values.

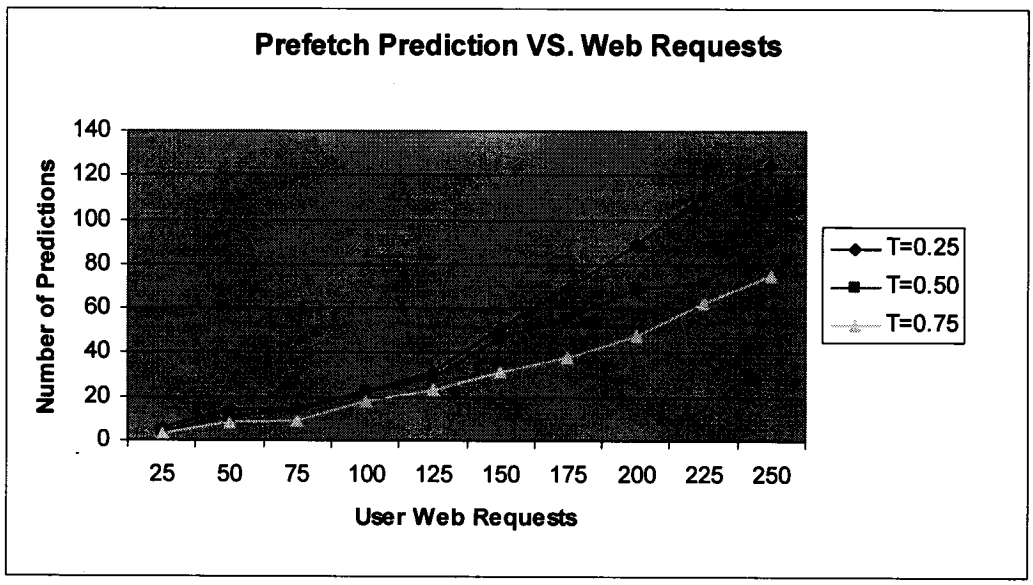

Figure 5.4: Prefetch predictions with different threshold value

5.2.6 Performance Of Prefetching

The definition of a successful prediction or performance is defined as the ratio of the total number of objects predicted by the prediction algorithm that are afterward requested by the user, to the total of predictions made by the prediction algorithm. We evaluated this experiment using different threshold values of 0.25, 0.50 and 0.75, and we obtained the following results.

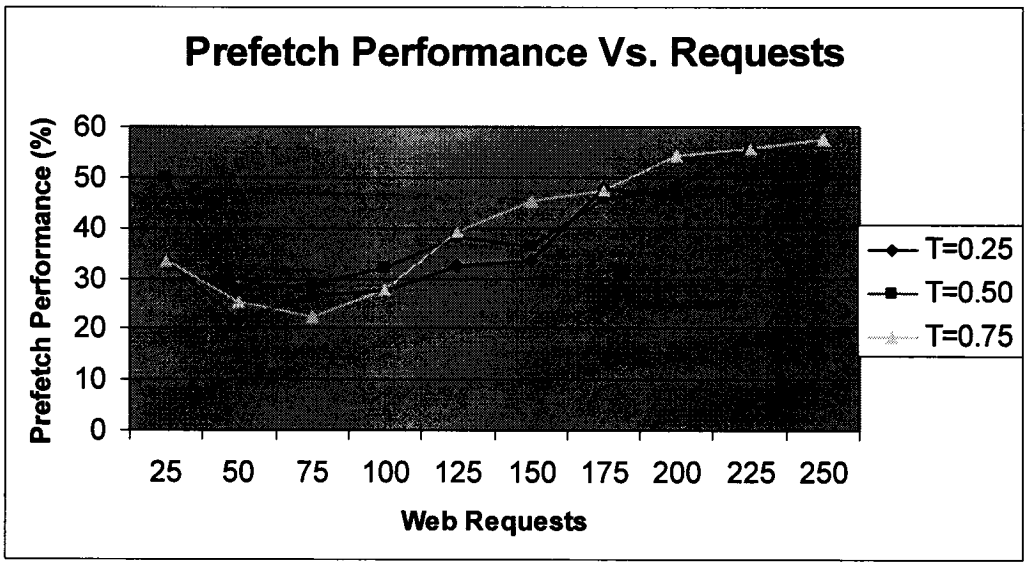

Figure 5.5: Prefetch hit ratio with different threshold value

As shown in Figure 5.5, it is clear that the higher threshold value produces a higher prefetch hit ratio while the lower threshold values produce a lower hit ratio.

5.2.7 Prefetch Waste Ratio

The definition of an unsuccessful prediction is the ratio of the total number of objects predicted by the prediction algorithm that are afterward not demanded by the user, to the total number of predictions made by the prediction algorithm. We performed this experiment using different threshold values of 0.25, 0.50, and 0.75.

The results from Figure 5.6 show that a lower threshold value produces a higher waste ratio, and a higher threshold value produces a lower waste ratio.

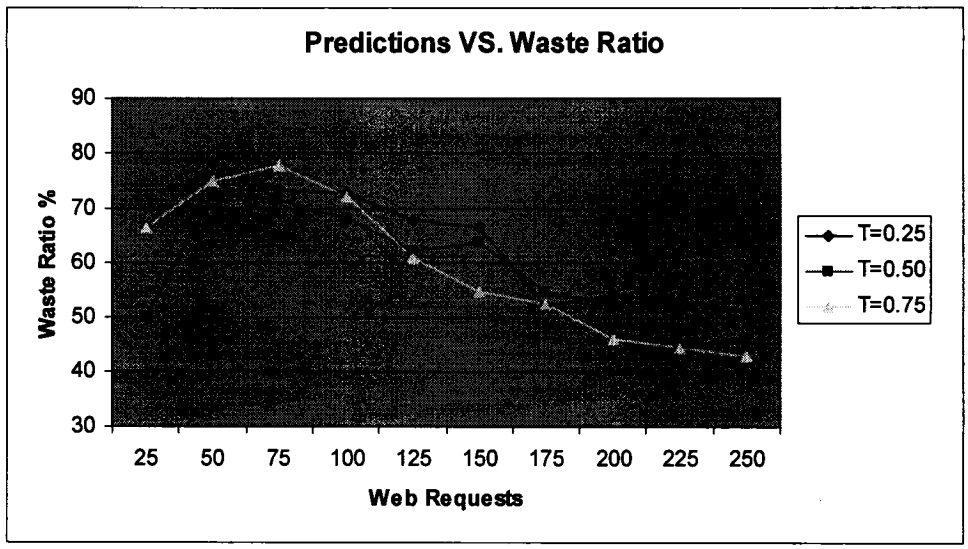

Figure 5.6: Prefetch waste ratio with different threshold value

The experimental results from Figures 5.4, 5.5 and 5.6, show that the lower threshold value produces a higher prediction rate and a higher waste ratio, but a lower prefetch hit ratio. The results from the above Figure show that there is a higher prefetch hit ratio, a lower waste ratio and a lower prediction rate with higher threshold value. There are more predictions with a lower threshold value, and it cached more undesired objects. In such a case, the cache may be quickly filled up with undesired objects.

5.3 Qualitative Comparison Between Proteus And Other Algorithms

This section briefly compares the different prefetch algorithms that we have discussed in detail in section 2.3, with our artificial intelligence multilayer neural network Proteus system. Although it is very difficult to compare these algorithms due to their various different features, Table 5.4 shows a comparison based on some of the prefetch aspects available in their documents.

	Proteus	**The Top 10** [29]	**Markov Graph** [38]	**Prediction by Partial Matching** [41]	**Adaptive Web Access Predictor** [35]	**Pre-fetching to Tolerate WWW Latency** [37]

Goal	Artificial Intelligence (Multilayer NN)	Popularity of the web document (Server Site)	Markov Modeling	Markov model + context	Neural Nets Based Predictive	Neural Nets Based Predictive
Personalized	YES	NO	NO	NO	NO	NO
Content Based	YES	NO	NO	YES	NO	YES
Content Filtering	YES	NO	NO	NO	NO	NO
History Based	NO	YES	YES	YES	YES	NO
Mobile Device (Mobility)	YES	NO	NO	NO	NO	NO
Server Based	Both Server and client	YES	YES	YES	NO	YES
Incorporate with Cache System	YES	NO	N/A	N/A	NO	NO
Infrastructure independency	NO	NO	N/A	N/A	N/A	YES
Cost	Medium	Very Low	Medium Higher	Higher	Higher	Medium

Table 5.4: Qualitative comparison between Proteus and other prefetch strategies

Most of the above terms are related to the prefetch features shown in column one of Table 5.4 and are already discussed in different sections of Chapter 2 and Chapter 3. We added four new terms (features) into column one with respect to this scope, such as goal, personalization, mobility and infrastructure dependency. The first row of the above table refers to the goal of the prefetch prediction algorithm, while the second row of the table signifies whether the prediction system is personalized or sharable.

The 3rd row compares the content-based feature of the algorithms. Content-based algorithms are more powerful than history-based algorithms because the content of the links help the user in the selection of the web objects or links. The 4th row of the table is

web content filtering. This is a unique feature of Proteus, which allows users to set up a clean Internet environment and enforces acceptable usage policies governing which URL can or cannot be preloaded. Web content filtering can block objectionable content such as pornography, racism, violence or criminal behaviour while preserving access to required Internet information resources.

The 5th row of the table compares whether the system is history based or not. History based prediction has some disadvantages such as it only works with previously visited sites or pages and not for new pages or sites. As shown in Table 5.4, the 6th row compares the mobility feature, which is not present in any other systems except Proteus.

The 7th row of the table refers to the requirements of a server based environment, indicating whether the proposed algorithm works only in a dedicated server mode (sharable) or not. Proteus can work as server based at the server end or in client mode. Being incorporating with the cache system (row 8) is a very important feature that provides better performance than individual systems. Proteus possesses this ability. As for the other proposed systems, this feature is either not available or could not be found in the system documentation. Infrastructure independence is also an important feature for a Proxy system due to the recent trends of network infrastructure changes such as security, wireless networks, etc.

The last row of this table is cost, and it is usually measured in terms of time and space. Time represents the processing time to complete a full cycle of a prefetch process, and space represents the quantity of information storage required or how much memory is required. Proteus obtains a better result than other Markov based prediction algorithms.

5.4 Performance Comparison Between Proteus And Other Algorithm

Table 5.5 briefly compares the performance of different prefetch algorithms with our artificial intelligence multilayer neural network Proteus system. We have collected the performance data from different corresponding research papers. To make the performance comparison, we considered two different matrices: prefetch hit ratio and traffic generation. From Table 5.5 it is show that Proteus, [35], and [37] obtained better

prefetch hit ratios than others. [35] claimed that they obtained a 60%-86% hit ratio which is the highest among the three, but they did not mention traffic generation. The prefetch hit ratios of Proteus and [37] are almost the same, but Proteus generates less traffic than [37]. We believe Proteus can achieve more than a 60% hit ratio if the system is trained for a considerable time.

	Proteus	**The Top 10** [29]	**Markov Graph** [38]	**Prediction by Partial Matching** [41]	**Adaptive Web Access Predictor** [35]	**Pre-fetching to Tolerate WWW Latency** [37]
Goal	Artificial Intelligence (Multilayer NN)	Popularity of the web document	Markov Modeling	Markov model + context	Neural Nets Based Predictive	Neural Nets Based Predictive
Prefetch Hit Ratio	57%	40%	NA**	57%	60%-86%	60%
Traffic Generation	12.8%	10%	NA**	18 %	NA*	20%

Table 5.5: Performance comparison between Proteus and other prefetch strategies.

* Author of [35] is not provided any data for traffic generates

** To evaluate the performance of [38] experimental result they used different matrices rather than prefetch hit ratio and traffic generation. They show in their result the variation of average file accesses time with prefetch threshold value and lock head window size.

5.5 Research Discussion:

Prefetching as web engineering is a fairly new technique. This technology is not used due to the drawbacks of present algorithms, such a lack of control over prefetching documents, or the system not sufficient being sufficiently intelligent to adequately learn user interests for prediction. Therefore, users can experience significant delays due to unwanted document downloads.

In this research, we present an integrated approach. Proteus is used for web caching and prefetching, which is based on artificial intelligence, and is used for the reduction of the

Internet access delay experienced by the Internet user community. We are very confident about our approach, since it addresses all of the above concerns, as follows.

- The AIMNN-based prefetching approach we use is based on well-defined threshold values for prediction. As a result, it preloads only a small number of appropriate documents. Our prefetch approach does not produce any additional traffic higher than 13 %.
- Our personalized artificially intelligent multilayer neural network based prediction algorithm emphasizes the user's web interest; it prefetches only the documents that are important to the user. With this approach, we considered both static and dynamic user web behaviors to ensure that the user's interests are accurately reflected in the prefetch documents.
- In our AIMNN prefetching approaches, we have used a content filtering option, which prevents the download of unwanted documents, a beneficial and unique feature.

In addition of resolving the above concerns, Proteus has several additional advantages that make it a useful tool for reducing Internet delay.

- Proteus uses a combined technique of caching and prefetching, resulting in a system that outperforms individual systems. It increases the hit ratio by approximately 26% with the integration of caching and prefetching techniques.
- Since Proteus is personalized, the chance of privacy or security violations is nonexistent.
- The mobility feature provides the user with the ability to move both caching and prefetching systems with a mobile computer, allowing the client to use this system anywhere or on any network.
- Since Proteus is browser independent, it allows the user to use any popular web browser for their browsing purposes. Also, Proteus can be installed on any platform since it is platform independent.

- Proteus is very user friendly due to easy configuration and management features. It also possesses multilingual support capabilities.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In our research, we present a context-specific artificially intelligence Proteus approach for improving the user's Internet experience by serving content from its local cache, or by preloading the user's predicted documents prior to the user requesting them. In this approach, we have used an artificially intelligence neural network algorithm for web caching, prefetching and web cache removal. The network learns the users' web interests or behaviours and then makes predictions based on these behaviours for future use. With this technique, the system relies on the keywords of URL anchor text to predict the user's future requests. This approach also prefetches the necessary documents whose URLs have not yet been visited by the user. It also provides new functionality such as content filtering and personalization, which are not available in other systems. This approach has self-learning capability and excellent adaptability to a change in the user's web interest. Analyzing our experimental observations, we conclude the following:

- The Proteus approach improves the performance of a user's Internet experience by implementing caching and prefetching simultaneously. Our experimental results shows that our system achieves an approximately 42% hit rate when combined caching and prefetching, compared to only 34% using caching alone.
- Prefetch experimental results show that AIMNN Proteus can achieve a prefetch hit ratio of approximately 57% and a waste ratio approximately 13%.
- The Proteus filter option provides content filtering, blocking unwanted documents such as pornography, racism, advertisements, etc.

6.2 Future Work

Future work will be focused on the three following aspects:

- The first step is to construct a prefetch module incorporating a thesaurus increasing the prefetch hit ratio.
- The second step is to correct the prediction and reduce the cost of processing, design a prefetch module with the capability to consider phrases or pairs of words,

instead of a single keyword model. In a single keyword model, it is necessary to run the prediction process for every single keyword if it there is more than one keyword for a single link.

- The third step is to construct a new caching strategy for dynamic web content since the amount of dynamic content is gradually increasing.

Glossary

Back propagation: Back propagation is a process by which the perceptron neural network is "trained" to produce good responses for a set of input patterns. The perceptron network is sometimes called a "back-prop" network.

Bias: Neural network input is proportional to the amount that incoming neural activations must exceed in order for a neuron to fire.

Browser: A browser that obtains web content for consumption by a human user or an application on the same computer. Therefore, the browser represents the final destination of web objects. The most common examples of browsers are Mozilla Firefox, Microsoft Internet Explorer and Netscape Navigator.

Cacheable: If a web object is allowed to store a copy of the response and return this copy on subsequent requests. Also, we call a request cacheable if it can be satisfied by a cached response. The request for a cacheable object may or may not be cacheable depending on the HTTP request headers.

Caching: Storing a copy of the data close to the consumer of the data to allow faster data access. Generally the term *caching* mainly refers to storing web responses closer to the web client requesting the contents. Another benefit of caching is that communication with the place where the data was originally stored is reduced. Examples of caches are browser caches and Back propagation: Back propagation is a process by which the perceptron neural network is "trained" to produce good responses for a set of input patterns. The perceptron network is sometimes called a "back-prop" network.

Bias: Neural network input is proportional to the amount that incoming neural activations must exceed in order for a neuron to fire.

Browser: A browser that obtains web content for consumption by a human user or an application on the same computer. Therefore, the browser represents the final destination of web objects. The most common examples of browsers are Mozilla Firefox, Microsoft Internet Explorer and Netscape Navigator.

Cacheable: If a web object is allowed to store a copy of the response and return this copy on subsequent requests. Also, we call a request cacheable if it can be satisfied by a cached response. The request for a cacheable object may or may not be cacheable depending on the HTTP request headers.

Caching: Storing a copy of the data close to the consumer of the data to allow faster data access. Generally the term *caching* mainly refers to storing web responses closer to the web client requesting the contents. Another benefit of caching is that communication with the place where the data was originally stored is reduced. Examples of caches are browser caches and proxy caches.

Cache hit: A request that can be satisfied using a cached response.

Cache misses: When a request cannot be satisfied using a cached response.

Client: An application program that requests service from another application or service from a server, which typically executes on a different host. For example, a web browser such as Internet Explorer and a proxy server are both web clients.

Connectivity: The connectivity of a neural network is defined as the amount of interaction in a system, the structure of the weights in a neural network, or the relative number of edges in a graph.

Document: The web document is defined as a collection of objects that are displayed as a single entity in a browser. An HTML object and all its embedded images constitute a single document.

Dynamic object: Dynamic objects are generated by the server for each time it processes a request for this object, as opposed to serving the object from a file.

Epoch: Epoch is defined as a complete presentation of the training set to the network during training.

Hit rate: The cache hit ratio of a request that a component such as a web proxy satisfies, using cached objects to the total number of requests that arrive at this component.

HTML: Hypertext Markup Language is a language used to encode documents on the Web.

HTTP: Hypertext Transfer Protocol is used in the Web to transfer objects between servers and clients.

IP: Internet Protocol is the protocol used to connect multiple data networks to the Internet.

IP address: Each host on the Internet has at least one 32-bit-long unique Internet address identifying the host. Typically, IP addresses are written in decimal dotted notation where each decimal number represents 8 bits in the address.

Latency: Latency is the elapsed time between the moment a client issues a request and the moment it receives the response. Sometimes a distinction is drawn between download latency and download time, in which case the latter includes the time for the client to receive the entire response, whereas the former includes only the time to receive the first byte of the response.

Learning algorithms: Neurons whose inputs are fed from the outside world.

Learning rule: The algorithm used for modifying the connection strengths, or weights, in response to training patterns while training is being carried out.

Monte-Carlo method: The Monte-Carlo method provides approximate solutions to a variety of mathematical problems by performing statistical sampling experiments on a computer.

Multilayer-perceptron (MLP): This is a type of feed forward neural network that is an extension of the perceptron in that it has at least one hidden layer of neurons. The layers of the neural network are updated by starting at the inputs and ending with the outputs. The neuron computes a weighted sum of the incoming signals, to yield a net input, and passes this value through its sigmoidal activation function to yield the neuron's activation value. Unlike the perceptron, an MLP can solve linearly inseparable problems.

Neural Network (NN): A network of neurons that are connected through synapses or weights. The neuron performs a simple calculation that is a function of the activations of the neurons that are connected to it. Through feedback mechanisms and/or the nonlinear output response of neurons, the network as a whole is capable of performing extremely complicated tasks, including universal computation and universal approximation. There are three different classes of neural networks are exists such as feedforward, feedback, and recurrent neural networks, which differ in the degree and type of connectivity that they possess.

Neuron: A neuron is a very simple computational unit that performs a weighted sum on incoming signals, adds a threshold or bias term to this value to yield a net input, and maps this last value through an activation function to compute its own activation.

Output neuron: An output of neuron is defined ad within a neural network whose outputs is the result of the network.

Perceptron: The neural network capable of simple pattern recognition and classification tasks. This is composed of three layers where signals only pass forward from nodes in the input layer to nodes in the hidden layer and finally out to the output layer.

Sigmoid function: An S-shaped function that is often used as an activation function in a neural network.

Threshold: A quantity added to (or subtracted from) the weighted sum of inputs into a neuron, which forms the neuron's net input. Intuitively, the net input (or bias) is proportional to the amount that the incoming neural activations must exceed in order for a neuron to fire.

Training set: A neural network is trained using a training set. A training set comprises information about the problem to be solved as input stimuli.

Weight: The strength of a synapse (or connection) between two neurons. Weights may be positive (excitatory) or negative (inhibitory).

Bibliography

[1] Abdulmotaleb El-Saddik1, Carsten Griwodz1, Ralf Steinmetz1 "Exploiting User Behaviour in Prefetching WWW Documents" International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services'98, Oslo, Norway Sept. 08-11, 1998

[2] Aggarwal, A., and Rabinvich, M. (1998). Performance of replication schemes on the Internet. Technical Report HA6177000-981030-01-TM, AT&T Labs.

[3] Aggarwal, M., Wolf, J., and Yu, P. (1999). Caching on the World Wide Web. IEEE Transactions on Knowledge and Data Engineering, 11:95-107.

[4] Anick, P. (2003). Using Terminological Feedback for Web Search refinement - a Log-Based Study. In Proc. of SIGIR 2003, Toronto, Canada. 88-95.

[5] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In Proceedings of IEEE Conference on Parallel and Distributed Information Systems, December 1996.

[6] Arlitt, M., Friedrich, R., and Jin, T. (1999).

[7] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in Web client access patterns: Characteristics and caching implications. World Wide Web: Special Issue on Characterization and Performance Evaluation.

[8] Berners-Lee, T., Fielding, R., and Neilsen, H. 1996. RFC 1945

[9] Bestavros, A (1995). Using speculation to reduce the server load and service time on the WWW. In proceedings of the 4th ACM International Conference on information and knowledge management, pp 403-410.

[10] Bestavros, A (1996). Speculative data dissemination and service to reduce server load, network traffic and service time in distributed system. In proceedings of the 12th international conference on data engineering, pp 180-189.

[11] Cater, J. P. Successfully Using Peak Learning Rates of 10 (and greater) in Back-Propagation Networks with the Heuristic Learning Algorithm. In Proceedings of the IEEE International Conference on Neural Networks, pages 645-651. San Diego, CA, 1987.

[12] Philip K. Chan. A non-invasive learning approach to building Web user pro_les. In Proceedings of WebKDD'99, pages 7{12, San Diego, August 1999. Revised version in Springer LNCS Vol. 1836.

[13] Cheng-Yue Chan and Ming-Syan Chen – Web Caching Replacement by integrating Caching and Prefetching, ACM, 2002

[14] Colleen Kehoe, Gaurav Aggarwal, Jim Pitkow, Juan D. Rogers, and Kate Sutton,. Results of GVU's tenth World Wide Web user survey Georgia Institute of Technology, Atlanta, GA: http://www.gvu.gatech.edu/user surveys/survey-1998-10/tenthreport.html, May 1999.

[15] C. R. Cunha and C. F. B. Jaccoud. Determining WWW user's next access and its application to prefetching. In Proc. of the International Symposium on Computers and Communication'97, July 1997.

[16] Cockburn, A., McKenzie, B., and Jason Smith, M. (2002). Pushing Back: Evaluating a New Behaviour for the Back and Forward Buttons in Web Browsers. Int. J. Human-Computer Studies, 57: 397-414.

[17] DAVISON, B. D. 2001. A Web caching primer. IEEE Internet Compute. 5, (July), 38–45.

[18] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L. Leach, P., and Berners-Lee, T. (1999). RFC 2616: HyperText Transfer Protocol, HTTP/1.1-http:///www.ietf.org/rfc/rfc2616.txt.

[19] Gary, C. G. and Cheriton, D.R. (1989). Leases: An efficient fault tolerant mechanism for distributed file cache consistence. In Procedings of the 12th ACM Symosium on Operating System Principles, pp. 202-210.

[20] H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press. 1995

[21] Hong, J. I., Heer, J., Waterson, S., and Landay, J. A. (2001). Webquilt: A Proxy-Based Approach to Remote Web Usability Testing. ACM Transactions on Information Systems, 19(3): 263-285.

[22] HOWARD, J, KAZAR, M., MENEES, S., NICHOLS, D., SATYANARA YANAN, M. SIDE BOTHAM R., AND WEST, M. 1988. Scale and performance in a distributed file system. ACM Trans. Comput.syst. 6, 1 (Feb, 1988), 51-81.

[23] 1996 by Ingrid Russell, Department of Computer Science, University of Hartford

[24] "Engineering Web caching consistence" JIAN YIN, LOREENZO ACVISI, and MIKE DAHLIN, university of Texas at Austin, and ARUN IVENAGAR, IBM T.J. Watson Research centre.

[25] Jianliang Xu "Caching and Prefetching for Web Content Distribution1" Computing in Science and Engineering archive, Volume 6 , Issue 4 (July 2004) table of contents Pages: 54 - 59, Year of Publication: 2004, ISSN:1521-9615

[26] Jin, S., and Bestavros, A. (2000). Source and characteristics of web temporal locality. In proceedings of the 8th International Symposium on Modeling, Analysis and simulation of computer and Telecommunication system, pp. 28-36.

[27] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in Proceedings of the USENIX Symposium on Internet Technology and Systems, pp. 13-22, Dec. 1997.

[28] "A Top-10 Approach to Prefetching on the Web", Evangelos P. Markatos, Catherine E. Chronaki., Institute of Computer Science (ICS), Foundation for Research and Technology (1996).

[29] Markatos, E. P., and Chronaki, C. E. (1998). A top 10 approach for prefetching the web. In Processing of the INET conference

[30] Milic-Frayling, N., Jones, R., Rodden, K., Smyth, G., Blackwell, A., and Sommerer, R. (2004). Smartback: Supporting Users in Back Navigation. In Proc. of WWW 2004, New York, NY. 63-71.

[31] Michael Rabinovich and Oliver Spatscheck, ISBN 0-201-61570-3

[32] Tom M. Mitchell, Sridhar Mahadevan, and Louis I. Steinberg. LEAP: A learning apprentice for VLSI design. In Proceedings of the Ninth International Joint Conference on Arti_cial Intelligence, Los Angeles, CA, August 1985.

[33] Venkata N. Padmanabhan and Je_rey C. Mogul. Using predictive prefetching to improve World Wide Web latency. Computer Communication Review, 26(3):22{36, July 1996. Proceedings of SIGCOMM '96.

[34] Tauscher, L. and Greenberg, S. (1997). How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems.

[35] Wen Tian, Ben Choi, and Vir Phoha, "An Adaptive Web Cache Access Predictor Using Neural Network," Developments in Applied Artificial Intelligence, IEA/AIE 2002, Lecture Notes in Artificial Intelligence, Vol. 2358, pp. 450-459, 2002.

[36] YIN, J, ALVISI L., DAHLIN, M., AND LINC 1998. Using leases to support server-driven consistency in a large scale system. In proceedings of the Eighteenth international conference on distributed computing system (May 1998).

[37] Tamer I. Ibrahim and Cheng-Zhong Xu , Neural Nets Based Predictive Pre-fetching to Tolerate WWW Latency, Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202

[38] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. In Proc. of ACM SIGCOMM Computer Comm. Review, July 1996.

[39] S. Schechter, M. Krishnan, and M. Smith. Using path profiles to predict HTTP requests. In Proccedings of the 7th International World Wide Web Conference. Also appeared in Computer Networks and ISDN Systems, 20(1998), pages 457— 467.

[40] Michael Zhen Zhang and Qiang Yang. Model-based predictive prefetching. In Proceedings of the 2nd International Workshop on Management of In-formation on the Web | Web Data and Text Mining (MIW'01), Munich, Germany, September 2001.

[41] Themistoklis Palpanas and Alberto Mendelzon. Web Prefetching Using Partial Match Prediction. In Proceedings of the Fourth International Web Caching Workshop (WCW99), San Diego, CA, March 1999.

[42] Fan, L., Cao, P., Lin, W., and Jacobson, Q. (1999). Web prefetching between low bandwidth client and proxies: Potential and performance. In proceedings of SIGMETRICS conference, pp. 178-187.

[43] Zona Research. "The economic impacts of unacceptable Web site download speeds", white paper, 1999. Can be found at from http://www.zonaresearch.com/deliverables/white papers/wp17/index.htm.

[44] Wlliams, S, Abrams, M., Standridge, C.R., Abdulla, G., and Fox, E.A. (1996). Removal policies in network cache for World-Wide Web document. In proceedings of the ACM SIGCOMM Conference, PP. 293-305.